



**IBM Cloud**  
**Watson Assistant**  
Product guide

## **Edition notices**

This PDF was created on 2023-03-17 as a supplement to *Watson Assistant* in the IBM Cloud docs. It might not be a complete set of information or the latest version. For the latest information, see the IBM Cloud documentation at <https://cloud.ibm.com/docs/watson-assistant>.

# Welcome to the new Watson Assistant

Welcome to the documentation for the new Watson Assistant!

The new Watson Assistant experience, focused on using **actions** to build customer conversations, is designed to make it simple enough for *anyone* to build a virtual assistant. Building, testing, publishing, and analyzing your assistant can all now be done in one simple and intuitive interface.

- New **navigation** provides a workflow for building, previewing, publishing, and analyzing your assistant.
- Each assistant has a **home page** with a task list to help you get started.
- Build conversations with **actions**, which represent the tasks you want your assistant to help your customers with. Each action contains a series of steps that represent individual exchanges with a customer.
- A new way to **publish** lets you review and debug your work in a draft environment before going live to your customers.
- Use a new suite of **analytics** to improve your assistant. Review which actions are being completed to see what your customers want help with, determine if your assistant understands and addresses customer needs, and decide how can you make your assistant better.
- Explore our [interactive demo site](#) to learn how Watson Assistant can be used to build powerful, scalable experiences for your users.

For more information about the new experience, see [FAQs about the new IBM Watson Assistant experience](#).

Visit [Getting started with Watson Assistant](#) for a tutorial series on building in the new experience.

## Switching the experience

---

You can easily switch back and forth between the new experience and the classic experience. However, the new experience provides a simplified user interface, an improved deployment process, and access to the latest features. For more information, see [Migrating to the new experience](#). If you do need to switch, see [Switching between new and classic experiences](#).

## Using Watson Assistant

---

Watson Assistant can be deployed as a managed cloud service or can be installed on premises. This documentation only applies to a managed cloud service or IBM Cloud Pak® for Data version 4.6 or later, and describes how to use the product regardless of how it is deployed. Information that applies exclusively to one deployment type is denoted by the appropriate tag:

- IBM Cloud Pak for Data **IBM Cloud Pak® for Data** version 4.6 or later for installed instances.
- IBM Cloud **IBM Cloud** for managed instances that are hosted by IBM Cloud.

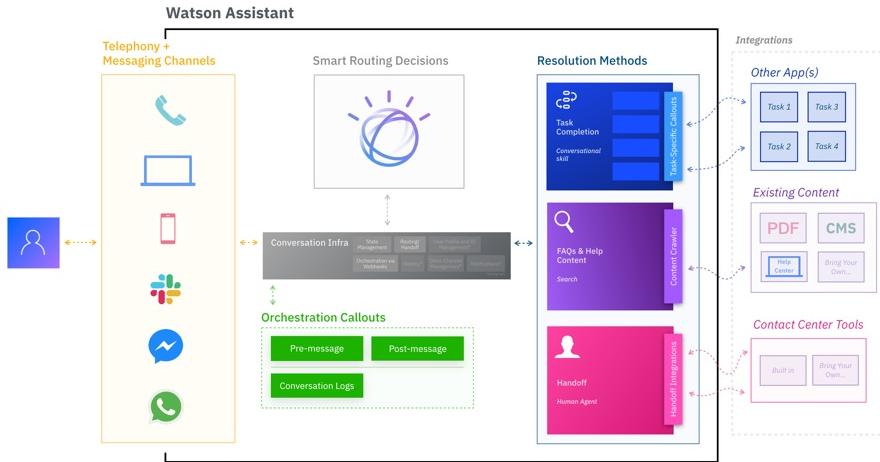
# About Watson Assistant

Use IBM Watson® Assistant to build your own branded live chatbot into any device, application, or channel. Your chatbot, which is also known as an *assistant*, connects to the customer engagement resources you already use to deliver an engaging, unified problem-solving experience to your customers.

<b>Create AI-driven conversational flows</b>	Your assistant uses industry-leading AI capabilities to understand questions that your customers ask in natural language. It uses machine learning models that are custom-built from your data to deliver accurate answers in real time.
<b>Embed existing help content</b>	You already know the answers to customer questions? Put your subject matter expertise to work. Add a search integration with IBM Watson® Discovery to give your assistant access to corporate data collections that it can mine for answers.
<b>Connect to your customer service teams</b>	If customers need more help or want to discuss a topic that requires a personal touch, connect them to human agents from your existing service desk provider.
<b>Bring the assistant to your customers, where they are</b>	Configure one or more built-in integrations to quickly publish your assistant on popular social media platforms such as Slack, Facebook Messenger, Intercom, or WhatsApp. Turn the assistant into a member of your customer support call center team, where it can answer the phone and address simple requests so its human teammates can focus on more nuanced customer needs. Make your assistant the go-to help resource for customers by adding it as a chat widget to your company website. If none of the built-in integrations fit your needs, use the APIs to build your own custom app.
<b>Track customer engagement and satisfaction</b>	Use built-in metrics to analyze logs from conversations between customers and your assistant to gauge how well it's doing and identify areas for improvement.

## How it works

This diagram illustrates how IBM Watson® Assistant delivers an exceptional, omnichannel customer experience:



Customers interact with the assistant through one or more of these channels:

- A web chat that you embed in your company website and that can transfer complex requests to a customer support representative.
- An existing social-media messaging platform, such as Slack, Facebook Messenger, or WhatsApp
- A phone call or text message
- A custom application that you develop, such as a mobile app or a robot with a voice interface

The **assistant** receives a message from a customer and sends it down the appropriate resolution path.

If you want to preprocess incoming messages, this is where you can use webhooks to inject logic that calls an external service that can process the messages before the assistant routes them. Likewise, you can process responses from the assistant before they are returned to the customer.

The assistant chooses the appropriate resolution from among these options:

- An **action** interprets the customer's message further, then directs the flow of the conversation. The action gathers any information it needs to respond or perform a transaction on the customer's behalf.
- A **search integration** uses existing FAQ or other curated content that you own to find relevant answers to customer questions.
- If a customer wants more personalized help or wants to discuss a sensitive subject, the assistant can connect the customer with someone from your support team through the web chat or phone integration.

To see how Watson Assistant is helping enterprises cut costs and improve customer satisfaction today, read the [Independent study finds IBM Watson Assistant customers can accrue \\$23.9 million in benefits](#) blog on ibm.com.

Read more about these implementation steps by following these links:

- [Building your assistant](#)
- [Publishing and deploying your assistant](#)

## **Browser support**

---

The Watson Assistant application requires the same level of browser software as is required by IBM Cloud. For more information, see IBM Cloud [Prerequisites](#).

For information about the web browsers that are supported by the web chat integration, see [Browser support](#).

## **Language support**

---

Language support by feature is detailed in [Supported languages](#).

# Getting started with Watson Assistant

The new Watson Assistant experience, focused on using **actions** to build customer conversations, is designed to make it simple enough for *anyone* to build a virtual assistant. Building, testing, publishing, and analyzing your assistant can all now be done in one simple and intuitive interface.

- New **navigation** provides a workflow for building, previewing, publishing, and analyzing your assistant.
- Each assistant has a **home page** with a task list to help you get started.
- Build conversations with **actions**, which represent the tasks you want your assistant to help your customers with. Each action contains a series of steps that represent individual exchanges with a customer.
- A new way to **publish** lets you review and debug your work in a draft environment before going live to your customers.
- Use a new suite of **analytics** to improve your assistant. Review which actions are being completed to see what your customers want help with, determine if your assistant understands and addresses customer needs, and decide how can you make your assistant better.

For more information about the new experience, see [FAQs about the new IBM Watson Assistant experience](#).

## Tutorial

---

This series of blog articles provides a tutorial to help you get started:

- [Plan it](#)
- [Part I: The build guide](#)
- [Part II: Refine your assistant](#)
- [Part III: Test and deploy](#)
- [Part IV: preview, draft, publish, live](#)

These starting points in the documentation also help you get started:

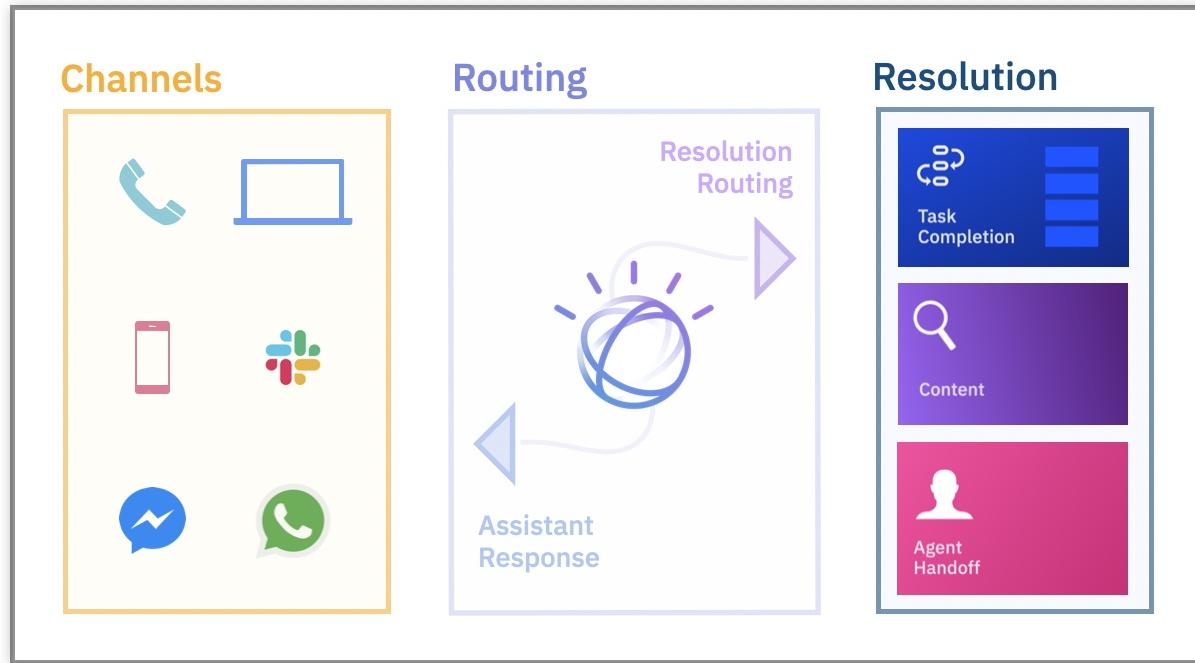
- [About Watson Assistant](#)
- [Planning your assistant](#)
- [Overview: Editing actions](#)
- [Building actions from a template](#)

# Planning your assistant

Before you start to build an assistant, it's important to think about your overall goals and plan how you're going to start.

Your assistant can use different resolution methods to help your customers with their requests. Customers access the assistant through channels that you choose and configure. The following diagram shows the structure of an assistant after you build it:

## Your first assistant



Consider these planning steps and make key decisions up front to keep you on track as you build.

### 1. Select an initial channel

Before you decide which specific topics to build into your assistant, you must first decide where to deploy your assistant so that customers can easily find it. For example, you can embed the assistant in your company website or add it to a messaging platform such as Facebook, Slack, or WhatsApp. The primary channels that you can use to communicate with your customers are your website and the phone.

If you choose to use [web chat](#) to communicate with customers on your website, decide on which of your web pages you want the assistant to appear. To start, you might identify the pages where your customers most frequently ask questions from your customer service team.

If you have an existing interactive voice response (IVR) system with a branching structure ("press 1 for billing, press 2 for payments"), you might choose the [phone integration](#) as your initial channel. You can integrate Watson Assistant with your existing system to automate the IVR experience so customers can talk with your assistant over the phone.

Understanding where you will deploy the assistant before you begin can help you author the right types of answers for a given channel or platform. For more information about ways to deploy an assistant, see [Adding integrations](#).

### 2. Pick your assistant's domain of expertise

Decide which general domain of expertise you want your assistant to cover (for example, billing support or scheduling appointments). To make an informed decision, review any support call logs that you have access to or ask your customer service representatives. After you choose a domain, be sure that it aligns with a channel that you can control and change. For example, don't choose to automate billing support questions if you're unable to add the web chat client to the billing web pages.

After you select a domain, you can decide which specific questions or tasks the assistant will help customers with. Start small. Pick one or a handful of customer issues that will deliver the highest value to start. It might be valuable for your assistant to answer a simple question that is asked all the time. Or maybe there's a task, such as scheduling appointments, that you can offload to the assistant to tackle incoming customer requests.

Choose a narrow set of user goals first. If you start small and choose the goals with the highest impact first, you'll have room and time to grow the expertise of your assistant. After your assistant is live, the built-in metrics of active user conversations help you understand what your customers are asking about, how well your assistant is able to meet their needs, and what to focus on next.

### **3. Choose the tone and language of your assistant**

---

Before you build your assistant, it can also be helpful to choose the tone with which your assistant communicates. Is your assistant an optimist or a pessimist, a shy intellectual type, or an upbeat sidekick? Write conversations that reflect your assistant's personality. Don't overdo it by sacrificing usability for the sake of keeping your assistant in character. Strive to present a consistent tone and attitude.

Never misrepresent the assistant as being a human. If users believe that the assistant is a person, then find out it's not, they are likely to distrust it. In fact, some US states have laws that require chatbots to identify themselves as chatbots.

Decide whether and how you want to handle more than one spoken language. For more information about ways to approach language support, see [Adding support for global audiences](#).

### **4. Connect to content sources**

---

The answer to common questions might already be documented somewhere in your organization's technical information. Plan whether you want to connect the assistant to existing content sources by taking an inventory of relevant help content (for example, product information, knowledge articles, FAQs) available to your customers.

You can give your assistant access to this information by adding a [search integration](#) to your assistant. The search integration uses IBM Watson® Discovery to return smart answers to natural language questions.

### **5. Plan your handoff strategy**

---

Finally, prevent customers from hitting dead ends by [escalating conversations to a human agent](#) if the assistant isn't able to resolve a customer's question or problem. As part of your planning, figure out what your escalation strategy is going to be. This strategy will vary depending on the deployment channel you choose.

If you deploy to your website, you have three options for escalating to a human agent:

- Inline
- Email address
- Phone number

If you opt for the inline approach, you can directly escalate to a human agent in your existing contact center tool without forcing the user to leave the web chat widget. This approach also provides the agent with the full context of the conversation. To use either the email address or phone number approach, provide the user with the agent's email address or phone number. These approaches are simple to set up, but they can be more disconnected experiences for your customers.

If you deploy by using the phone integration, you can reach a human agent only by transferring the phone call to someone who can help.

## **Start building an assistant**

---

If you're ready to start building an assistant, see [Editing actions](#) for more information.

# Release notes

## Release notes for Watson Assistant

---

Find out what's new in IBM Watson® Assistant.

This topic describes the new features, changes, and bug fixes in each release of the product. For more information about changes in the web chat integration, see the [Web chat release notes](#).

### 16 March 2023

#### New algorithm version Latest (20 Dec 2022) provides improved irrelevance detection

A new algorithm version is available. The **Latest (20 Dec 2022)** version includes a new irrelevance detection implementation to improve off-topic detection accuracy.

Improvements include:

- Relevant user inputs are expected to get higher confidence, so they are less likely to be considered irrelevant or require clarification
- Irrelevance detection is improved in the presence of direct entity references
- Irrelevance detection is more stable across small changes to input
- Intent detection is more stable regarding occurrence of numerics, such as postal codes
- For German-language assistants, intent detection is more robust in the presence of umlauts

This algorithm was first introduced as the **Beta** version in June 2022. Since then, support for more languages has been added. This algorithm version was stabilized in December 2022 with minor enhancements since that time.

With this new release, the June 1, 2022 version is now labeled as **Previous (01 Jun 2022)**. The oldest release labeled as **01 Jan 2022** is no longer available for training. As of now, the new **Beta** version has the same behavior as the **Latest (20 Dec 2022)** version. Updates to the **Beta** version will be released soon.

For more information, see [Algorithm version and training](#).

### 10 March 2023

#### Dialog session variables now available in Preview

If you are using dialog in the new experience, you can now see session variables for dialog when debugging in Preview. For more information, see [Variable values in Preview](#).

### 6 March 2023

#### Improvements to algorithm version beta

Improvements to the current *Beta* algorithm version include:

- Relevant examples are expected to get higher confidence

- For Spanish-language assistants, intent detection is improved in the presence of direct entity references
- Intent detection is more stable regarding occurrence of numerics, such as postal codes
- Intent detection now accounts for fuzzy closed entity mentions
- For German-language assistants, intent detection is more robust in the presence of umlauts

For more information, see [Algorithm version and training](#).

## 3 March 2023

### **Adding and using multiple environments**

Each assistant has a draft and live environment. For Enterprise plans, you can now add up to three environments as a staging area to test your assistant before deployment. You can build content in the draft environment and test versions of your content in the extra environments. For more information, see [Adding and using multiple environments](#).

### **Confirmation to return to previous action**

If a customer digresses and changes to a new topic, assistants now ask a "yes or no" confirmation question that the customers want to return to the previous action. Previously, the assistant returned to the previous action without asking. New assistants are set to use this confirmation by default. For more information, see [Confirmation to return to previous topic](#).

## 1 March 2023

### **Unrecognized requests group names**

This change improves the group names for unrecognized requests. For groups with examples phrased in the form of question, the group name can be more indicative of a question rather than a request. For more information, see [Use unrecognized requests to get action recommendations](#).

## 23 February 2023

### **Private variables excluded from logs**

Private context variables are no longer saved in logs or sent to external services using log webhooks. Private variables are any values stored inside the following objects:

- `context.integrations.*.private` (accessible from actions as `system_integrations.*.private`)
- `context.integrations.*.$private`
- `context.skills.*.user_defined.private`
- `context.skills.*.user_defined.$private`
- `context.private`
- `context.$private`

## 16 February 2023

## Improvements to setting variable values

When you use **Set variable values** on an action step:

- The available choices now match by type. For example, if you want to set a date variable, the choices are limited to other date variables. Previously, all variables of all types were listed as choices.
- You can set a scalar value for each variable type. For example, you can set a specific date for a date variable or set a specific number for a number variable.

For more information, see [Storing a value in a session variable](#).

## Confirmation and free text response types setting default

The *Confirmation* and *Free text* response types are now set to **Always ask for this information** by default. For more information, see [Skipping steps, always asking steps, or never asking steps](#).

## 13 February 2023

### Response variations

In actions, you can add *response variations* so that your assistant can respond to the same request in different ways. You can choose to rotate through the response variations sequentially or in random order. For more information, see [Adding variations](#).

### Microsoft Teams integration

A [Microsoft Teams integration](#) is now available to connect your assistant with the people, content, and tools that your business or community needs to chat, call, and collaborate.

## 3 February 2023

### Action conditions (beta)

An action condition is a boolean test, based on some runtime value; the action executes only if the test evaluates as true. This test can be applied to any variable. By defining action conditions, you can do things such as control user access to actions or create date-specific actions. This is a beta feature that is available for evaluation and testing purposes. For more information, see [Adding conditions to an action](#).

## 2 February 2023

### Detect trigger words (beta)

Use the **Trigger word detected** action to add words or phrases to two separate groups. The first group connects customers with an agent, when it's important for a customer to speak with a live agent rather than activate any further actions. The second group shows customers a customizable warning message, used to discourage customers from interacting with your assistant in unacceptable ways, such as using profanity. This action is included with all new assistants created as of this date. This is a beta feature that is available for evaluation and testing purposes. For more information, see [Detecting trigger words](#).

## Changes to unrecognized requests algorithm

In **Analyze**, the **Recognition** page lets you view groups of similar unrecognized requests. You can use the requests as example phrases in new or existing actions to address questions and issues that aren't being answered by your assistant. With this release, the criteria for grouping the requests is relaxed for customers with lesser amounts of data. Also, the group names have been improved with better grammar and to be more representative of the requests. For more information, see [Use unrecognized requests to get action recommendations](#).

## 1 February 2023

### Actions templates updated with new design and new choices

The actions template catalog has a new design that lets you select multiple templates at the same time. It also has new and updated templates, including starter kits you can use with external services such as Google and HubSpot. For more information, see [Building actions from a template](#).

## 26 January 2023

### Display formats for variables

In **Global settings** for actions, **Display formats** lets you specify the display formats for variables that use date, time, numbers, currency, or percentages. You can also choose a default locale to use if one isn't provided by the client application. This lets you make sure that the format of a variable that's displayed in the web chat is what you want for your assistant. For example, you can choose to have the output of a time variable appear in HH:MM format instead of HH:MM:SS. For more information, see [Display formats](#).

## 18 January 2023

### Algorithm version stability improvement

As of this date, the **Latest (01 Jun 2022)** and **Beta** algorithm versions now have more stable behavior across retrained models, in the presence of overlapping entities (the same entity value belonging to more than one entity type). Previously, when there were overlapping entities definitions, confidences could differ across different retraining. With this improvement, you can expect to see similar confidences. For more information, see [Algorithm version and training](#).

## 12 January 2023

### Autocorrection setting for actions

The **Global settings** for actions now include an **Autocorrection** setting. *Autocorrection* fixes misspellings that users make in their requests. The corrected words are used to match to an action.

While the setting is new, the autocorrection feature was already used automatically by all English-language assistants. Autocorrection is also available in French-language assistants, but is disabled by default. The autocorrection setting isn't available for any other languages. The new setting lets you disable or enable autocorrection if necessary.

For more information, see [Autocorrecting user input](#).

### Improved experience when setting a variable value

The dropdown list for setting a variable value within an action step has a new organization. The new list is intended to provide an improved experience.

## 11 January 2023

### Algorithm version 01-Jun-2022 uses enhanced intent detection by default

As of this date, the algorithm version **Latest (01-Jun-2022)** now uses enhanced intent detection by default. Before this change, some skills that did not include a specific algorithm version selection inadvertently used **Previous (01-Jan-2022)**. You can notice small changes in intent detection behavior when changes are made to an assistant that previously didn't have enhanced intent detection enabled. For more information, see [Algorithm version and training](#).

## 6 December 2022

### Updated expression methods

The following new and updated methods are available in expressions:

- The [Array.joinToArray\(\)](#) method now supports a new boolean parameter you can use to specify that the data type of values from the input array should be preserved in the returned array.
- The new [String.toJson\(\)](#) method parses a string containing JSON data and returns a JSON object or array. This method is supported in both actions and dialog.

## 5 December 2022

### Live integrations deleted in assistants created before June 24, 2022

For assistants created before June 24, 2022, using the new Watson Assistant user experience, the live integrations for these assistants were mistakenly deleted during a software upgrade. These integrations should now be restored. If you are still experiencing issues, please contact IBM support.

### Unsupported HTML removed from text responses in channel integrations

HTML tags (except for links) are now automatically removed from text responses that are sent to the Facebook, WhatsApp, and Slack integrations, because those channels do not support HTML formatting. HTML tags are still handled appropriately in channels that support them (such as the web chat) and stored in the session history.

## 2 December 2022

### Pause response type

Use a *Pause* response to have your assistant wait for a specified interval before displaying the next response. This pause might be to allow time for a request to complete, or simply to mimic the appearance of a live agent who might pause

between responses. The pause can be of any duration from 1 to 10 seconds. For more information, see [Pause response](#).

## 17 November 2022

### Use unrecognized requests to get action recommendations

In **Analyze**, the new **Recognition** page lets you view groups of similar unrecognized requests. You can use the requests as example phrases in new or existing actions to address questions and issues that aren't being answered by your assistant. For more information, see [Use unrecognized requests to get action recommendations](#).

## 15 November 2022

### Journeys

Beginning with web chat version 6.9.0, you can now create *journeys* to guide your customers through tasks they can already complete on your website. A journey is an interactive, multipart response that can combine text, video, and images, presented in sequence in a small window superimposed over your website.

Journeys are available as a beta feature. For more information, see [Guiding customers with journeys](#).

## 10 November 2022

### Dynamic options

Within the options customer response, you can use the **dynamic** setting to generate the list when you need to ask questions that are potentially different each time and for each customer. You need to set up a list variable as the source of the options. For more information, see [Dynamic options](#).

### Extension inspector

You can use the new extension inspector in the action editor **Preview** pane to debug problems with custom extensions. The extension inspector shows detailed information about what data is being sent to and returned from an external API. For more information, see [Debugging failures](#).

## 3 November 2022

### Never ask a step

There may be some situations where you need a step to never ask a question because you anticipate there might be redundant questions in the conversation. A new setting, **Never ask**, is now available for any step that expects a customer response. For more information, see [Skipping steps, always asking steps, or never asking steps](#).

### Action notes

You can now add free-form notes to each action. Within each action, you can use **Action notes** to add a description, documentation, comments, or any other annotations to help you keep track of your work as you build an action. For more information, see [Using the action editor](#).

### Variable values in Preview

Viewing action variables in Preview has been improved. Now you can see the history of all action variables, rather than one action at a time. For more information, see [Variable values](#).

## 21 October 2022

### Algorithm version updates

The algorithm version setting for both actions and dialog now includes three choices: beta, latest, and previous. For more information, see [Algorithm version and training](#).

## 12 October 2022

### now(String timezone) method output includes time zone offset

The string returned from the `now(String timezone)` method now includes the time zone offset (such as `-05:00`). The new format is `yyyy-MM-dd HH:mm:ss 'GMT'XXX` (where `XXX` represents the time zone offset). This change enables accurate time zone computations when used with other date and time methods such as `before`, `after`, and `reformatDateTime`.

If you have an existing action or dialog that depends on the previous format, you can adapt it by reformatting the output using `now(timezone).reformatDateTime('yyyy-MM-dd HH:mm:ss')`.

For more information, see [Expression language methods for actions](#).

## 23 September 2022

### Upload an image as a preview background

On the **Preview** page, you can now upload an image of your organization's website as a background. For more information, see [Previewing and sharing your assistant](#).

## 16 September 2022

### Session ID information on Analyze page

Session ID information for conversations is now displayed on the Conversations tab of the Analyze page. You can also filter customer conversation data by the session ID. From the Conversations tab of the Analyze page, use the Keyword filter to search by session ID. For more information, see [Filtering conversations](#).

The ability to filter on session ID has limited support for conversations that occurred before this feature release. For all conversations that occurred before 16 September 2022, you can filter only by a single session ID at a time.

## 9 September 2022

## New operators available for building conditions

Several new operators are available for building conditions in your actions. The free text response type now has the `is any of` and `is none of` operators available. For more information, see [Operators](#).

## Copy actions to other assistants

You can copy an action from one assistant to another. When you copy an action, references to other actions, variables, and saved responses are also copied. For more information, see [Managing actions](#).

## Filter variables and saved responses by name

You can now find variables and saved responses more easily. On the Actions page, you can filter variables you created or saved responses you added. Click the search icon, then enter a search string. Your list of variable or saved responses filters to match what you enter.

## 1 September 2022

### Conditioning on days of the week

You can now condition a step on days of the week. This feature is available with the `date` response type and the `Current date` built-in variable.

For example, you might [define a customer response](#) in step 1 with the date response type. When the customer responds to that step, they choose a date. You can then condition a later step on whether the date that the customer chose is Wednesday.

## New operators available for building conditions

Several new operators are available for building conditions in your actions. The free text response type now has the `contains`, `does not contain`, `matches`, and `does not match` operators available. For more information, see [Operators](#).

## Extensions support for arrays

Custom extensions now support passing arrays as parameters and accessing arrays in response variables. For more information, see [Calling a custom extension](#).

## 26 August 2022

### New filter on the Analyze page

You can now filter customer conversation data by the `Greet customer` system action. From the Conversations tab of the Analyze page, open the Actions filter and select **Greet customer**. For more information, see [Filtering conversations](#).

## Filter actions by name

You can now find actions more easily. On the Actions page, you can filter actions by name. Click the search icon, then enter a search string. Your list of actions filters to match what you enter.

## 12 August 2022

## **Actions templates**

When creating actions, you can choose a template that relates to the problem you're trying to solve. Templates help tailor your actions to include items specific to your business need. The examples in each template can also help you to learn how actions work. Actions templates include features such as intents, entities, condition-based responses, synonyms, response validations, and agent fallback. For more information, see [Building actions from a template](#).

## **Channel name variable**

The `Channel.name` integration variable lets you add step conditions using these channels: web chat, phone, SMS, WhatsApp, Slack, or Facebook Messenger. For more information, see [Adding conditions to a step](#).

## **11 August 2022**

### **Algorithm version options available in more languages**

Algorithm version options are now available in Arabic, Czech, and Dutch. This allows you to choose which Watson Assistant algorithm to apply to your future trainings. For more information, see [Algorithm version and training](#).

## **9 August 2022**

### **New API methods**

The v2 API now supports new **Environments** and **Releases** methods:

- **Environments:** Retrieve information about the environments associated with an assistant.
- **Releases:** Retrieve information about the releases (versions) that have been published for an assistant, and assign an available release to an environment.

For more information, see the v2 [API Reference](#).

## **5 August 2022**

### **Initial value of session variables**

You can now set the initial value of a session variable to an expression. For more information, see [Creating a session variable](#).

### **Uploading intents**

If you created intents in the classic Watson Assistant experience, you can migrate your intents to actions in the new Watson Assistant experience. This can provide a helpful starting point when you are ready to start building actions in the new experience. For more information, see [Uploading intents as actions](#).

## **19 July 2022**

### **Changes to publishing and environments**

You can now publish versions of your content without assigning to the live environment, allowing you to make continuous updates before customers see it in production. Also, the formerly separate pages for your draft and live environments now appear as tabs on a single *Environments* page, from which you can set up unique configurations for building and testing in the draft environment, and for your customers in the live environment. For more information, see the [Publishing overview](#).

### Logs reader role

Identity and Access Management now includes a new service role, **Logs Reader**, which lets you grant access to Analytics without assigning the Manager role. Use Logs Reader in combination with the Reader or Writer role to provide access to the Analytics page. For more information, see [Managing access](#).

## 15 July 2022

### Segment extension

The Segment extension is now available for Enterprise plans. With this extension, you can use [Segment](#) to capture and centralize data about your customers' behavior, including their interactions with your assistant. For more information, see [Sending events to Segment](#).

### New expression property

You can now use the `.literal` property to return the exact response that a customer specifies. This property is helpful if a customer uses a synonym of an option, and you want your assistant to respond with the exact phrase they specified. To set this property, click the **Set variable values** icon and assign a session variable to the step variable. Add the `.literal` property to the step variable. Use the session variable in the assistant's response to display the customer's input.

For example, suppose you have an option called `plant` that has `fern` as a synonym. A customer might say `buy a fern`. In this case, you can use the `.literal` property so the assistant's response uses the customer's input. Your assistant might respond, `Great! I see you want to buy a fern.`

## 11 July 2022

### Ability to duplicate an action

You can duplicate an action to reuse information in a new action. When you duplicate an action, the new action includes everything except example phrases. Click the overflow menu on the action you want and select **Duplicate**.

### New demo site

Explore our [interactive demo site](#) to learn how Watson Assistant can be used to build powerful, scalable experiences for your users.

## 24 June 2022

### Algorithm version options available in more languages

Algorithm version options are now available in Chinese (Traditional), Japanese, and Korean. This allows you to choose which Watson Assistant algorithm to apply to your future trainings. For more information, see [Algorithm version and training](#).

## 16 June 2022

### Algorithm version

Algorithm version allows you to choose which Watson Assistant algorithm to apply to your future trainings. For more information, see [Algorithm version and training](#).

### Algorithm beta version (2022-06-10)

Algorithm beta version (2022-06-10) includes a new irrelevance detection algorithm to improve off-topic detection accuracy. Utterances with similar meanings are expected to have more similar confidences in comparison to previous irrelevance detection algorithms. For example, the training utterance `please suggest route from times square` has 100% confidence at runtime. Currently in IBM Cloud, the utterance `please suggest route from central park` gets a low confidence and could be flagged as irrelevant. With beta version (2022-06-10), the same utterance is expected to be predicted correctly with a ~46% confidence.

## 3 June 2022

### Extensions support for importing API document with unsupported methods

When building a custom extension, you can now import an API document even if it contains operations with required array parameters, which are not supported. The unsupported operations are automatically disabled, but this does not affect other operations. (Previously, the entire API document was rejected if it contained unsupported operations.) For more information, see [Building a custom extension](#).

## 27 May 2022

### Support for custom extensions and dialog in Actions preview panel

You can now view your entire assistant from the **Actions preview** panel, including custom extensions and dialog. This allows you to have a complete view of how an action is working. For more information about previewing actions, see [Reviewing and debugging your actions](#).

## 19 May 2022

### Sign out due to inactivity setting

Watson Assistant now uses the **Sign out due to inactivity setting** from Identity & Access Management (IAM). IBM Cloud account owners can select the time it takes before an inactive user is signed out and their credentials are required again. The default is 2 hours.

An inactive user will see two messages. The first message alerts them about an upcoming session expiration and provides a choice to renew. If they remain inactive, a second session expiration message appears and they will need to log in again.

For more information, see [Setting the sign out due to inactivity duration](#).

## 12 May 2022

## Ability to upload and download example phrases and upload saved customer responses

You can now upload and download example phrases from **Customer starts with** at the start of an action. This can be useful if you have a large number of example phrases and don't want to define them one by one. For more information, see [Adding more examples](#).

You can also now upload saved customer responses from the **Saved responses** page. For more information, see [Uploading saved customer responses](#).

The ability to upload example phrases and saved customer responses is also helpful if you're using the classic Watson Assistant and want to migrate your intents and entities to the new Watson Assistant. For more information, see [Migrating intents and entities](#).

## 5 May 2022

### Success/failure variable for extensions

Each call to a custom extension now returns a `ranSuccessfully` response variable, which you can use to check the success or failure of the call. For more information, see [Checking success or failure](#).

## 28 April 2022

### Definitive calculation for abandoned actions

Abandonment is now definitively calculated for your actions. On the **Analytics** page, actions are no longer considered Ongoing in the action completion analysis. An action is considered abandoned if it was not completed after 1 hour of inactivity and doesn't meet the criteria for any other incompletion reason (escalated to agent, started a new action, or stuck on a step). This change applies only to actions data after April 26, 2022. For more information about action incompletion, see [Reasons for incompletion](#).

### Managing operations in extensions

When you add a custom extension to an assistant, you can now choose which operations and response properties will be available to actions. For more information, see [Adding an extension to your assistant](#).

## 21 April 2022

### Ability to duplicate a step

You can now duplicate a step so you don't have to re-create variable settings and customizations. Duplicating a step is helpful when you need to add a step similar to a previous step, but with minor modifications. For more information about how to duplicate a step, see [Duplicating a step](#).

### Markdown supported in action editor

The action editor now supports basic Markdown syntax. As you type, the action editor renders the Markdown so you can see the content as your customers will when they interact with the assistant.

## 5 April 2022

### Dialog feature available

The dialog feature is available. If you have a dialog-based assistant that was built using the classic Watson Assistant, you can now migrate your dialog skill to the new Watson Assistant experience. For more information, see [Migrating to the new experience](#).

## 28 March 2022

### New service desk support reference implementation

You can use the reference implementation details to integrate the web chat with the Kustomer service desk. For more information, see [Adding service desk support](#).

## 18 March 2022

### Custom extensions

If you need to integrate your assistant with an external service that has a REST API, you can now build a custom extension by importing an OpenAPI document. Your assistant can then send requests to the external service and receive response data it can use in the conversation. For example, you might use an extension to interact with a ticketing or customer relationship management (CRM) system, or to retrieve real-time data such as mortgage rates or weather conditions.

For more information about custom extensions, see [Building a custom extension](#) and [Calling a custom extension](#).

### Confirmation customer response type

The confirmation customer response type is now available. Use this response type when a customer's response must be either Yes or No. For more information, see [Confirmation](#).

### Search integration highlights text in browser

Search results in Watson Assistant include a link. Now, when a customer clicks the link, search results are highlighted in their browser so it's easier for them to see the relevant content. This feature is supported on Chromium browsers, including Google Chrome and Microsoft Edge.

## 24 February 2022

### Regex customer response type

The regex customer response type is now available. Use this response type to capture a value that must conform to a particular pattern or format, such as an email address or telephone number. For more information, see [Regex](#).

## 17 February 2022

## **Adding users from the Manage menu**

If you want to collaborate with others on your assistants, you can now quickly add users with Administrator and Manager access from the **Manage** menu in your assistant. For more information, see [Adding users from the Manage menu](#).

## **Preview page share link**

When you use the **Copy link to share** button to share your assistant, the shared assistant now mirrors the Preview page. If you share the assistant with a colleague, they are able to see the assistant with any customizations that you made on the Preview page.

## **10 February 2022**

### **Links in assistant responses can be configured to open in a new tab**

When you build an action, your assistant responses can include links. If you're using web chat, you can now control whether the link opens in a new tab. To enable a link to open in a new tab, select **Open link in new tab** from the **Insert link** configuration window. For more information, see [Adding assistant responses](#).

## **9 February 2022**

### **All instances now default to new experience**

All new instances of Watson Assistant now direct users to the new product experience by default.

Watson Assistant has been completely overhauled to simplify the end-to-end process of building and deploying a virtual assistant, reducing time to launch and enabling nontechnical authors to create virtual assistants without involving developers. For more information about the new Watson Assistant, and instructions for switching between the new and old experiences, see [Welcome to the new Watson Assistant](#).

If you would like to send us feedback on the new experience, please use [this form](#).

## **3 February 2022**

### **Customize the Preview page background**

You can now change the background of the **Preview** page to one of your organization's web pages so you can preview and test your assistant from a customer's perspective. For more information, see [Previewing and sharing your assistant](#).

### **Add a type to session variables**

When you create a session variable, you can now assign a type to the variable. For more information, see [Creating a session variable](#). After a type is assigned to a variable, you can set more explicit conditions on that variable. Previously, you were able to check only whether session variables were `defined` or `not defined`. With variable types, you can create conditions based on the type of the variable (for example, `account balance < 100` or `departure date is after today`). For more information, see [Operators](#).

### **Create saved customer responses**

You can now create saved customer responses. There might be some questions that your assistant needs to ask in different

steps and actions. For example, a banking assistant might have different actions that ask for a customer's account number. Instead of building the same response over and over, you can create a saved customer response and reuse it across steps in multiple actions. For more information, see [Saving and reusing customer responses](#).

## 13 January 2022

### New setting for options customer response type

In actions, a new **List options** setting allows you to enable or disable the options customer response from appearing in a list. This can be useful to prevent a phone integration from reading a long list of options to the customer. As part of this change, all customer response types now have a **Settings** icon. **Allow skipping** has moved from **Edit Response** and is now found in the new settings. For more information, see [Collecting information from your customers](#).

## 24 December 2021

### Apache Log4j security vulnerability updates

Watson Assistant upgraded to using Log4j version 2.17.0, which addresses all of the Critical severity and High severity Log4j CVEs, specifically CVE-2021-45105, CVE-2021-45046, and CVE-2021-44228.

## 3 December 2021

### Configure webhook timeout

From the **Pre-message webhook** and **Post-message webhook** configuration pages, you can configure the webhook timeout length from a minimum of 1 second to a maximum of 30 seconds. For more information, see [Extending your assistant with webhooks](#).

### User-based switching between Watson Assistant experiences

Previously, switching between the new Watson Assistant experience and the classic Watson Assistant experience was instance-based. For example, if a user switched from the classic experience to the new experience, all users of that Watson Assistant instance were switched to the new experience. Now, switching between the experiences is user-based. So, any user of a Watson Assistant instance can switch between the new and classic experiences, and other users of that Watson Assistant instance are not affected.

## 27 November 2021

### New API version

The current API version is now `2021-11-27`. This version introduces the following changes:

- The `output.text` object is no longer returned in `message` responses. All responses, including text responses, are returned only in the `output.generic` array.

## 12 November 2021

### Completion analytic information

On the **Analyze** page, the **How often** chart can now also show the percentage of complete actions. Use the icon in the upper right of the chart to toggle between a line chart that shows the percentage of complete actions and a bar chart that shows the number of complete and incomplete actions. For more information, see [Improving completion](#).

### Preview page update

The **Test integrations** panel no longer exists on the **Preview** page. You can manage your draft web chat channel from the **Preview** page. However, all other draft environment integrations are managed from the **Draft environment** page. For more information, see [Previewing and sharing your assistant](#).

## 4 November 2021

### Draft and Live Environment pages

Two pages, **Draft environment** and **Live environment**, help you to see how your channels and resolution methods are connected, both for testing/preview and for live deployment. The Draft environment page is new as of this release. The Live environment page was previously named Connect. For more information, see [Overview: Publishing and deploying your assistant](#).

### Add variables to links

When including a link in an assistant response, you can now access and use variables. In the URL field for a link, type a dollar sign (\$) character to see a list of variables to choose from.

## 25 October 2021

### Facebook and Slack integrations now available

The new Watson Assistant now includes integrations for [Facebook Messenger](#) and [Slack](#).

### Analytics for draft and live environments

The **Analyze** page now lets you see analytics data for either the draft or live environments. For more information, see [Use analytics to review your entire assistant at a glance](#).

## 7 October 2021

### The new Watson Assistant

The new Watson Assistant is now available! This new experience, focused on using **actions** to build customer conversations, is designed to make it simple enough for *anyone* to build a virtual assistant. Building, testing, publishing, and analyzing your assistant can all now be done in one simple and intuitive interface.

- New **navigation** provides a workflow for building, previewing, publishing, and analyzing your assistant.
- Each assistant has a **home page** with a task list to help you get started.
- Build conversations with **actions**, which represent the tasks you want your assistant to help your customers with. Each

action contains a series of steps that represent individual exchanges with a customer.

- A new way to **publish** lets you review and debug your work in a draft environment before going live to your customers.
- Use a new suite of **analytics** to improve your assistant. Review which actions are being completed to see what your customers want help with, determine if your assistant understands and addresses customer needs, and decide how can you make your assistant better.
- New [Top intents and top entities](#)

## 16 September 2021

### Enhanced intent detection for French, Italian, and Spanish dialog skills

The new intent detection model improves your assistant's ability to understand what customers want. This model is now available in dialog skills using French, Italian, and Spanish. For more information, see [Improved intent recognition](#).

### Change to the irrelevance detection option

As of this release, new English dialog skills no longer include the option to choose between the **Enhanced** or **Existing** irrelevance detection. By default, intent detection and irrelevance detection are paired like this:

- If you use the dialog skill options to choose enhanced intent detection, it is automatically paired with enhanced irrelevance detection.
- If you use the dialog skill options to choose existing intent detection, it is automatically paired with existing irrelevance detection.

For more information, see [Defining what's irrelevant](#) and [Improved intent recognition](#).

If necessary, you can use the [Update workspace API](#) to set your English-language assistant to one of the four combinations of intent and irrelevance detection:

- Enhanced intent recognition and enhanced irrelevance detection
- Enhanced intent recognition and existing irrelevance detection
- Existing intent recognition and enhanced irrelevance detection
- Existing intent recognition and existing irrelevance detection

For French, Italian, and Spanish, you can use the API to set your assistant to these combinations:

- Enhanced intent recognition and enhanced irrelevance detection
- Existing intent recognition and existing irrelevance detection

## 15 September 2021

### Dialog skill "Try it out" improvements

The **Try it out** pane now includes these changes:

- It now includes runtime warnings in addition to runtime errors.
- For dialog skills, the **Try it out** pane now uses the [React](#) UI framework similar to the rest of the Watson Assistant user

interface. You shouldn't see any change in behavior or functionality. As a part of the update, dialog skill error handling has been improved within the "Try it out" pane. This update was enabled on these dates:

- September 9, 2021 in the Tokyo and Seoul data centers
- September 13, 2021 in the London, Sydney, and Washington, D.C. data centers
- September 15, 2021 in the Dallas and Frankfurt data centers

## 13 September 2021

### Dialog skill "Try it out" improvements

For dialog skills, the **Try it out** pane now uses the [React](#) UI framework similar to the rest of the Watson Assistant user interface. You shouldn't see any change in behavior or functionality. As a part of the update, dialog skill error handling has been improved within the "Try it out" pane. This update was enabled on September 9, 2021 in the Tokyo and Seoul data centers. On September 13, 2021, the update was enabled in the London, Sydney, and Washington, D.C. data centers.

### Disambiguation feature updates

The dialog skill disambiguation feature now includes improved features:

- **Increased control:** The frequency and depth of disambiguation can now be controlled by using the `sensitivity` parameter in the [workspace API](#). There are 5 levels of sensitivity:

- `high`
- `medium_high`
- `medium`
- `medium_low`
- `low`

The default (`auto`) is `medium_high` if this option is not set.

- **More predictable:** The new disambiguation feature is more stable and predictable. The choices shown may sometimes vary slightly to enable learning and analytics, but the order and depth of disambiguation is largely stable.

These new features may affect various metrics, such as disambiguation rate and click rates, as well as influence conversation-level key performance indicators such as containment.

If the new disambiguation algorithm works differently than expected for your assistant, you can adjust it using the `sensitivity` parameter in the update workspace API. For more information, see [Update workspace](#).

## 9 September 2021

### Actions skill improvements

Actions skills now include these new features:

- **Change conversation topic:** In general, an action is designed to lead a customer through a particular process without any interruptions. In real life, however, conversations almost never follow such a simple flow. In the middle of a

conversation, customers might get distracted, ask questions about related issues, misunderstand something, or just change their minds about what they want to do. The **Change conversation topic** feature enables your assistant to handle these digressions, dynamically responding to the user by changing the conversation topic as needed. For more information, see [Changing the topic of the conversation](#).

- **Fallback action:** The built-in action, *Fallback*, provides a way to automatically connect customers to a human agent if they need more help. This action helps you to handle errors in the conversation, and is triggered by these conditions:
  - Step validation failed: The customer repeatedly gave answers that were not valid for the expected customer response type.
  - Agent requested: The customer directly asked to be connected to a human agent.
  - No action matches: The customer repeatedly made requests or asked questions that the assistant did not understand.

For more information, see [Set by assistant actions](#).

### Dialog skill "Try it out" improvements

For dialog skills, the **Try it out** pane now uses the [React](#) UI framework similar to the rest of the Watson Assistant user interface. You shouldn't see any change in behavior or functionality. As a part of the update, dialog skill error handling has been improved within the "Try it out" pane. This update will be implemented incrementally, starting with service instances in the Tokyo and Seoul data centers.

## 2 September 2021

### Deploy your assistant on the phone in minutes

We have partnered with [IntelePeer](#) to enable you to generate a phone number for free within the phone integration. Simply choose to generate a free number when following the prompts to create a phone integration, finish the setup, and a number is assigned to your assistant. These numbers are robust and ready for production.

### Connect to your existing service desks

We have added step-by-step documentation for connecting to [Genesys](#) and [Twilio Flex](#) over the phone. Easily hand off to your live agents when your customers require telephony support from your service team. Watson Assistant deploys on the phone via SIP, so most phone based service desks can easily be integrated via SIP trunking standards.

## 23 August 2021

### Intent detection updates

Intent detection for the English language has been updated with the addition of new word-piece algorithms. These algorithms improve tolerance for out-of-vocabulary words and misspelling. This change affects only English-language assistants, and only if the enhanced intent recognition model is enabled. (For more information about the enhanced intent recognition model, and how to determine whether it is enabled, see [Improved intent recognition](#).)

### Automatic retraining of old skills and workspaces

As of August 23, 2021, Watson Assistant enabled automatic retraining of existing skills in order to take advantage of updated algorithms. The Watson Assistant service will continually monitor all ML models, and will automatically retrain those models that have not been retrained within the previous 6 months. For more information, see [Automatic retraining of old](#)

[skills and workspaces](#).

## 19 August 2021

### **Actions preview now includes debug mode and variable values**

When previewing your actions, you can use **debug mode** and **variable values** to ensure your assistant is working the way you expect.

**Debug mode** allows you to go to the corresponding step by clicking on a step locator next to each message. It shows you the confidence score of top three possible action when the input triggers an action. You can also follow the step in the action editor along with the conversation flow.

**Variable values** shows you a list of the variables and their values of current action and the session variables. You can check and edit variables during the conversation flow.

## 17 August 2021

### **New service desk support reference implementation**

You can use the reference implementation details to integrate the web chat with the Oracle B2C Service service desk. For more information, see [Adding service desk support](#).

## 29 July 2021

### **Salesforce and Zendesk deployment changes**

The Salesforce and Zendesk integrations have been updated to use the [new chat history widget](#). The updated deployment process applies to all new deployments, including any redeployments of existing Salesforce and Zendesk connections. However, existing deployments are not affected and do not need to be modified or redeployed at this time.

### **Fallback value for session variables**

In action skills, you can now set a fallback value for session variables. This feature lets you to define a value for a session variable if a user-defined value isn't found. To learn more, see [Defining session variables](#).

## 16 July 2021

### **Logging API changes**

The internal storage and processing of logs has changed. Some undocumented fields or filters might no longer be available. (Undocumented features are not officially supported and might change without notice.)

### **New API version**

The current API version (v1 and v2) is now [2021-06-14](#). The following changes were made with this version:

- The `metadata` property of entities detected at run time is deprecated. For detailed information about detected system

entities, see the `interpretation` property.

- The data types of certain entity mentions are no longer automatically converted:
  - Numbers in scientific notation (such as `1E10`), which were previously converted to numbers
  - Boolean values (such as `false`), which were previously converted to booleans

These values are now returned as strings.

## 17 June 2021

### **Actions skill now generally available**

As of this release, the beta program has ended, and actions skills are available for general use.

An actions skill contains actions that represent the tasks you want your assistant to help your customers with. Each action contains a series of steps that represent individual exchanges with a customer. Building the conversation that your assistant has with your customers is fundamentally about deciding which steps, or which user interactions, are required to complete an action. After you identify the list of steps, you can then focus on writing engaging content to turn each interaction into a positive experience for your customer. For more information, see [Actions skill overview](#).

### **Date and time response types**

New to action skills, these response types allow you to collect date and time information from customers as they answer questions or make requests. For more information, see [Response types](#).

### **New built-in variables**

Two kinds of built-in variables are now available for action skills.

- **Set by assistant** variables include the common and essential variables `Now`, `Current_time`, and `Current_date`.
- **Set by integration** variables are `Timezone` and `Locale` and are available to use when connected to a webhook or integration.

For more information, see [Adding and referencing variables](#).

### **Universal language model now generally available**

You now can build an assistant in any language you want to support. If a dedicated language model is not available for your target language, create a skill that uses the universal language model. The universal model applies a set of shared linguistic characteristics and rules from multiple languages as a starting point. It then learns from training data written in the target language that you add to it. For more information, see [Understanding the universal language model](#).

## 3 June 2021

### **Log webhook support for actions and search skills**

The log webhook now supports messages exchanged with actions skills and search skills, in addition to dialog skills. For more information, see [Logging activity with a webhook](#).

## 27 May 2021

### Change to conversation skill choices

When adding skills to new or existing assistant, the conversation skill choices have been combined, so that you pick from either an actions skill or a dialog skill.

With this change:

- New assistants can use up to two skills, either actions and search or dialog and search. Previously, new assistants could use up to three skills: actions, dialog, and search.
- Existing assistants that already use an actions skill and a dialog skill together can continue to use both.
- The ability to use actions and dialog skills together in a new assistant is planned for 2H 2021.

## 20 May 2021

### Actions skill improvement

Actions now include a new choice, **Go to another action**, for what to do next in a step. This feature lets you call one action from another action, to switch the conversation flow to another action to perform a certain task. If you have a portion of an action that can be applied across multiple use cases you can build it once and call to it from each action. This new option is available in the **And then** section of each step. For more information, see [Deciding what to do next](#).

## 21 April 2021

### Preview button for testing your assistant

For testing your assistant, the new Preview button replaces the previous Preview tile in Integrations.

### New checklist with steps to go live

Each assistant includes a checklist that you can use to ensure you're ready to go live.

### Actions skill improvement

Actions now include currency and percentage response types.

### Learn what's new

The *What's new* choice on the help menu opens a list of highlighting recent features.

## 14 April 2021

### Actions skill improvement

Actions now include a free text response type, allowing you to capture special instructions or requests that a customer wants to pass along.

## 8 April 2021

## Deploy your assistant to WhatsApp - now generally available

Make your assistant available through WhatsApp messaging so it can exchange messages with your customers where they are. This integration, which is now generally available, creates a connection between your assistant and WhatsApp by using Twilio as a provider. For more information, see [Integrating with WhatsApp](#).

## Web chat home screen now generally available

Ease your customers into the conversation by adding a home screen to your web chat window. The home screen greets your customers and shows conversation starter messages that customers can click to easily start chatting with the assistant. For more information about the home screen feature, see [Configuring the home screen](#). The home screen feature is now enabled by default for all new web chat deployments. Also, you can now access context variables from the home screen. Note that initial context must be set using a `conversation_start` node. For more information, see [Starting the conversation](#).

## Connect to human agent response type allows more text

In a dialog skill, the response type `Connect to human agent` now allows 320 characters in the `Response when agents are online` and `Response when no agents are online` fields. The previous limit was 100 characters.

## Legacy system entities deprecated

In January 2020, a new version of the system entities was introduced. As of April 2021, only the new version of the system entities is supported for all languages. The option to switch to using the legacy version is no longer available.

## 6 April 2021

### Service API endpoint change

As explained in [December 2019](#), as part of work done to fully support IAM authentication, the endpoint you use to access your Watson Assistant service programmatically is changing. The old endpoint URLs are deprecated and **will be retired on 26 May 2021**. Update your API calls to use the new URLs.

The pattern for the endpoint URL changes from `gateway-{location}.watsonplatform.net/assistant/api/` to `api.{location}.assistant.watson.cloud.ibm.com/`. The domain, location, and offering identifier are different in the new endpoint. For more information, see [Updating endpoint URLs from watsonplatform.net](#).

- If your service instance API credentials show the old endpoint, create a new credential and start using it today. After you update your custom applications to use the new credential, you can delete the old one.
- For a web chat integration, you might need to take action depending on when and how you created your integration.
  - If you tied your deployment to a specific web chat version by using the `clientVersion` parameter and specified a version earlier than version 3.3.0, update the parameter value to use version 3.3.0 or later. Web chat integrations that use the latest or 3.3.0 and later versions will not be impacted by the endpoint deprecation.
  - If you created your web chat integration before May 2020, check the code snippet that you embedded in your web page to see if it refers to `watsonplatform.net`. If so, you must edit the code snippet to use the new URL syntax. For example, change the following URL:

```
$ <script src="https://assistant-web.watsonplatform.net/loadWatsonAssistantChat.js"></script>
```

The correct syntax to use for the source service URL looks like this:

```
$ src="https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssistantChat.js"
```

- If your web chat integration connects to a Salesforce service desk, then you must edit the API call that is included in the code snippet that you added to the Visualforce Page that you created in Salesforce. From Salesforce, search for *Visualforce Pages*, and find your page. In the `<iframe>` snippet that you pasted into the page, make the following change:

Replace: `src="https://assistant-integrations-{location}.watsonplatform.net/public/salesforceweb"` with a url with this syntax:

```
src="https://integrations.{location}.assistant.watson.appdomain.cloud/public/salesforceweb/{integration-id}/agent_application?version=2020-09-24"
```

From the Web chat integration Salesforce live agent setup page, find the *Visualforce page markup* field. Look for the `src` parameter in the `<iframe>` element. It contains the full URL to use, including the appropriate `{location}` and `{integration-id}` values for your instance.

- For a Slack integration that is over 7 months old, make sure the Request URL is using the proper endpoint.
  - Go to the [Slack API](#) web page. Click *Your Apps* to find your assistant app. Click *Event Subscriptions* from the navigation pane.
  - Edit the Request URL.

For example, if the URL has the syntax: `https://assistant-slack-{location}.watsonplatform.net/public/message`, change it to have this syntax:

```
https://integrations.{location}.assistant.watson.appdomain.cloud/public/slack/{integration-id}/message?version=2020-09-24
```

Check the *Generated request URL* field in the Slack integration setup page for the full URL to use, which includes the appropriate `{location}` and `{integration-id}` values for your instance.

- For a Facebook Messenger integration that is over 7 months old, make sure the Callback URL is using the proper endpoint.
  - Go to the [Facebook for Developers](#) web page.
  - Open your app, and then select *Messenger* > *Settings* from the navigation pane.
  - Scroll down to the *Webhooks* section and edit the *Callback URL* field.

For example, if the URL has the syntax: `https://assistant-facebook-{location}.watsonplatform.net/public/message/`, change it to have this syntax:

```
https://integrations.{location}.assistant.watson.appdomain.cloud/public/facebook/{integration-id}/message?version=2020-09-24
```

Check the *Generated callback URL* field in the Facebook Messenger integration setup page for the full URL to use, which includes the appropriate `{location}` and `{integration-id}` values for your instance.

- For a Phone integration, if you connect to existing speech service instances, make sure those speech services use credentials that were generated with the latest endpoint syntax (a URL that starts with `https://api.`

{location}.speech-to-text.watson.cloud.ibm.com/ ).

- For a search skill, if you connect to an existing Discovery service instance, make sure the Discovery service uses credentials that were generated with the supported syntax (a URL that starts with `https://api.{location}.discovery.watson.cloud.ibm.com/`).
- If you are using [Jupyter notebooks](#) to do advanced analytics, check your Jupyter notebook files to make sure they don't specify URLs with the old `watsonplatform.net` syntax. If so, update your files.
- No action is required for the following integration types:
  - Intercom
  - SMS with Twilio
  - WhatsApp with Twilio
  - Zendesk service desk connection from web chat

## 23 March 2021

### **Actions skill improvement**

Actions have a new toolbar making it easier to send feedback, access settings, save, and close.

## 17 March 2021

### **Channel transfer response type**

Dialog skills now include a channel transfer response type. If your assistant uses multiple integrations to support different channels for interaction with users, there might be some situations when a customer begins a conversation in one channel but then needs to transfer to a different channel. The most common such situation is transferring a conversation to the web chat integration, to take advantage of web chat features such as service desk integration. For more information, see [Adding a Channel transfer response type](#).

### **Intercom and WhatsApp integrations now available in Lite plan**

The integrations for Intercom and WhatsApp are now available in the Lite plan for Watson Assistant. For more information, see [Integrating with Intercom](#) and [Integrating with WhatsApp](#).

## 16 March 2021

### **Session history now generally available**

Session history allows your web chats to maintain conversation history and context when users refresh a page or change to a different page on the same website. It is enabled by default. For more information about this feature, see [Session history](#).

Session history persists within only one browser tab, not across multiple tabs. The dialog provides an option for links to open in a new tab or the same tab. See [this example](#) for more information on how to format links to open in the same tab.

Session history saves changes that are made to messages with the `pre:receive event` so that messages still look the same on rerender. This data is only saved for the length of the session. If you prefer to discard the data, set

```
event.updateHistory = false;
```

 so the message is rerendered without the changes that were made in the pre:receive event.

[instance.updateHistoryUserDefined\(\)](#) provides a way to save state for any message response. With the state saved, a response can be rerendered with the same state. This saved state is available in the `history.user_defined` section of the message response on reload. The data is saved during the user session. When the session expires, the data is discarded.

Two new history events, [history:begin](#) and [history:end](#) announce the beginning and end of the history of a reloaded session. These events can be used to view the messages that are being reloaded. The history:begin event allows you to edit the messages before they are displayed.

See this example for more information on saving the state of [customResponse](#) types in session history.

## Channel switching

You can now create a dialog response type to functionally generate a connect-to-agent response within channels other than web chat. If a user is in a channel such as Slack or Facebook, they can trigger a channel transfer response type. The user receives a link that forwards them to your organization's website where a connection to an agent response can be started within web chat. For more information, see [Adding a Channel transfer response type](#).

## 11 March 2021

### Actions skill improvement

Updated the page where you configure a step with an *Options* reply constraint. Now it's clearer that you have a choice to make about whether to always ask for the option value or to skip asking. For more information, see [Apply reply constraints](#).

## 4 March 2021

### Support for every language!

You now can build an assistant in any language you want to support. If a dedicated language model is not available for your target language, create a skill that uses the universal language model. The universal model applies a set of shared linguistic characteristics and rules from multiple languages as a starting point. It then learns from training data written in the target language that you add to it.

The universal model is available as a beta feature. For more information, see [Understanding the universal language model](#).

### Actions skill improvement

Now you can indicate whether or not to ask for a number when you apply a number reply constraint to a step. Test how changes to this setting might help speed up a customer's interaction. Under the right circumstances, it can be useful to let a number mention be recognized and stored without having to explicitly ask the customer for it. For more information, see [Applying reply constraints](#).

## 1 March 2021

## Introducing the **Enterprise** plan!

The Enterprise plan includes all of the market differentiating features of the Plus plan, but with higher capacity limits, additional security features, custom onboarding support to get you going, and a lower overall cost at higher volumes.

To have a dedicated environment provisioned for your business, request the *Enterprise with Data Isolation* plan. To submit a request online, go to <http://ibm.biz/contact-wa-enterprise>.

The Enterprise plan is replacing the Premium plan. The Premium plan is being retired today. Existing Premium plan users are not impacted. They can continue to work in their Premium instances and create instances up to the 30-instance limit. New users do not see the Premium plan as an option when they create a service instance.

For more information, see the [Pricing](#) page.

## Other plan changes

Our pricing has been revised to reflect the features we've added that help you build an assistant that functions as a powerful omnichannel SaaS application.

Starting on 1 March 2021, the Plus plan starts at \$140 per month and includes your first 1,000 monthly users. You pay \$14 for each additional 100 active users per month. Use of the voice capabilities that are provided by the *Phone* integration are available for an additional \$9 per 100 users per month.

The Plus Trial plan was renamed to Trial.

## SOC 2 compliance

Watson Assistant is SOC 2 Type 2 compliant, so you know your data is secure.

The System and Organization Controls framework, developed by the American Institute of Certified Public Accountants (AICPA), is a standard for controls that protect information stored in the cloud. SOC 2 reports provide details about the nature of internal controls that are implemented to protect customer-owned data. For more information, see [IBM Cloud compliance programs](#).

## 25 February 2021

### Search skill can emphasize the answer

You can configure the search skill to highlight text in the search result passage that Discovery determines to be the exact answer to the customer's question. For more information, see [Creating a search skill](#).

### Integration changes

The following changes were made to the integrations:

- The name of *Preview link* integration changed to *Preview*.
- The *Web chat* and *Preview* integrations are no longer added automatically to every new assistant.

The integrations continue to be added to the *My first assistant* that is generated for you automatically when you first create a new service instance.

### Message and log webhooks are generally available

The premessage, postmessage, and log webhooks are now generally available. For more information about them, see [Webhook overview](#).

## 11 February 2021

### The `user_id` value is easier to access

The `user_id` property is used for billing purposes. Previously, it was available from the context object as follows:

- v2: `context.global.system.user_id`
- v1: `context.metadata.user_id`

The property is now specified at the root of the `/message` request in addition to the context object. The built-in integrations typically set this property for you. If you're using a custom application and don't specify a `user_id`, the `user_id` is set to the `session_id` (v2) or `conversation_id` (v1) value.

### Digression bug fix

Fixed a bug where digression setting changes that were made to a node with slots were not being saved.

## 5 February 2021

### Documentation update

The phone and SMS with *Twilio* deployment documentation was updated to include instructions for migrating from Voice Agent with Watson. For more information, see [Integrating with phone](#) and [Integrating with SMS with Twilio](#).

## 27 January 2021

### German language improvements

A word decomposition function was added to the intent and entity recognition models for German-language dialog skills.

A characteristic of the German language is that some words are formed by concatenating separate words to form a single compound word. For example, "festnetznummer" (landline number) concatenates the words "festnetz" (landline) and "nummer" (number). When your customers chat with your assistant, they might write a compound word as a single word, as hyphenated words, or as separate words. Previously, the variants resulted in different intent confidence scores and different entity mention counts based on your training data. With the addition of the word decomposition function, the models now treat all compound word variants as equivalent. This update means you no longer need to add examples of every variant of the compound words to your training data.

## 19 January 2021

### The Phone and SMS with Twilio integrations are now generally available!

For more information, see:

- [Integrating with phone](#)
- [Integrating with SMS with Twilio](#)

#### **Preview link change**

When you create a preview link, you can now test your skill from a chat window that is embedded in the page. You can also copy the URL that is provided, and open it in a web browser to see an IBM-branded web page with the web chat embedded in it. You can share the URL to the public IBM web page with others to get help with testing or for demoing purposes. For more information, see [Testing your assistant](#).

#### **Import and export UI changes**

The label on buttons for importing skills changed from *Import* to *Upload*, and the label on buttons for exporting skills changed from *Export* to *Download*.

#### **Coverage metric change**

The coverage metric now looks for nodes that were processed with a node condition that includes the `anything_else` special condition instead of nodes that are named `Anything else`. For more information, see [Starting and ending the dialog](#).

## **15 January 2021**

#### **Use new webhooks to process messages!**

A set of new webhooks is available as a beta feature. You can use the webhooks to perform preprocessing tasks on incoming messages and postprocessing tasks on the corresponding responses. You can use the new log webhook to log each message with an external service. For more information, see [Webhook overview](#).

#### **New service desk support reference implementation**

You can use the reference implementation details to integrate the web chat with the NICE inContact service desk. For more information, see [Adding service desk support](#).

#### **Phone and SMS with Twilio integration updates**

The phone integration now enables you to specify more than one phone number, and the numbers can be imported from a comma-separated values (CSV) file. The *SMS with Twilio* integration no longer requires you to add your SMS phone number to the setup page.

## **6 January 2021**

#### **Import and export UI changes**

The label on buttons for importing intents and entities changed from *Import* to *Upload*. The label on buttons for exporting intents and entities changed from *Export* to *Download*.

## **4 January 2021**

## Dialog methods updates

Documentation and examples were added for the following supported dialog methods:

- `JSONArray.addAll(JSONArray)`
- `JSONArray.containsIgnoreCase(value)`
- `String.equals(String)`
- `String.equalsIgnoreCase(String)`

For more information, see [Expression language methods](#).

## 17 December 2020

### Accessibility improvements

The product was updated to provide enhanced accessibility features.

## 14 December 2020

### Increased Phone and SMS with Twilio integrations availability

These beta SMS and voice capabilities are now available from service instances that are hosted in Seoul, Tokyo, London, and Sydney.

### Improved JSON editor

The JSON editor in the dialog skill was updated. The editor now uses JSON syntax highlighting and allows you to expand and collapse objects.

### Connect to agent from actions skill

The actions skill now supports transferring a customer to an agent from within an action step. For more information, see [Deciding what to do next](#).

## 4 December 2020

### Introducing more service desk options for web chat

When you deploy your assistant by using the web chat integration, there are now reference implementations that you can use for the following service desks:

- Twilio Flex
- Genesys Cloud

Alternatively, you can bring your own service desk by using the service desk extension starter kit.

For more information, see [Adding service desk support](#).

### Autolearning has been moved and improved

Go to the *Analytics>Autolearning* page to enable the feature and see visualizations that illustrate how autolearning impacts

your assistant's performance over time. For more information, see [Empower your skill to learn automatically](#).

## Search from actions skill

The actions skill now supports triggering a search that uses your associated search skill from within an action step. For more information, see [Deciding what to do next](#).

## System entities language support change

The new system entities are now used by all skills except Korean-language dialog skills. If you have a Korean skill that uses the older version of the system entities, update it. The legacy version will stop being supported for Korean skills in March 2021. For more information, see [Legacy system entities](#).

## Disambiguation selection enhancement

When a customer chooses an option from a disambiguation list, the corresponding intent is submitted. With this latest release, a confidence score of 1.0 is assigned to the intent. Previously, the original confidence score of the option was used.

## Skill import improvements

Importing of large skills from JSON data is now processed in the background. When you import a JSON file to create a skill, the new skill tile appears immediately. However, depending on the size of the skill, it might not be available for several minutes while the import is being processed. During this time, the skill cannot be opened for editing or added to an assistant, and the skill tile shows the text **Processing**.

## 23 November 2020

### Deploy your assistant to WhatsApp!

Make your assistant available through WhatsApp messaging so it can exchange messages with your customers where they are. This beta integration creates a connection between your assistant and WhatsApp by using Twilio as a provider. For more information, see [Integrating with WhatsApp](#).

## 13 November 2020

### New coverage metric and enhanced intent detection model

The following features are available in service instances hosted in all data center locations except Dallas.

#### Introducing the coverage metric!

Want a quick way to see how your dialog is doing at responding to customer queries? Enable the new coverage metric to find out. The coverage metric measures the rate at which your dialog is confident that it can address a customer's request per message. For conversations that are not covered, you can review the logs to learn more about what the customer wanted. For the metric to work, you must design your dialog to include an *Anything else* node that is processed when no other dialog node intents are matched. For more information, see [Graphs and statistics](#).

#### Try out the enhanced intent detection model

The new model, which is being offered as a beta feature in English-language dialog and actions skills, is faster and more accurate. It combines traditional machine learning, transfer learning, and deep learning techniques in a cohesive model that is highly responsive at run time. For more information, see [Improved intent recognition](#).

## 3 November 2020

### Suggestions are now generally available

The Suggestions feature that is available for the web chat integration is generally available and is enabled by default when you create a new web chat integration. For more information, see [Showing more suggestions](#).

### New languages supported by the dialog analysis notebook

The *Dialog skill analysis notebook* was updated with language support for French, German, Spanish, Czech, Italian, and Portuguese. For more information, see [Analysis notebooks](#).

### Visit the learning center!

Click the **Learning center** link that is displayed in the header of the skill pages to find helpful product tours. The tours guide you through the steps to follow to complete a range of tasks, from adding your first intent to a dialog skill to enhancing the conversation in an actions skill. The **Additional resources** page has links to relevant documentation topics and how-to videos. You can search the resource link titles to find what you're looking for quickly.

## 29 October 2020

### System entity support changes

For English, Brazilian Portuguese, Czech, Dutch, French, German, Italian, and Spanish dialog skills only the new system entities API version is supported. For backward compatibility, both the `interpretation` and `metadata` attributes are included with the recognized entity object. The new system entity version is enabled automatically for dialog skills in the Arabic, Chinese, Korean, and Japanese languages. You can choose to use the legacy version of the system entities API by switching to it from the **Options>System Entities** page. This settings page is not displayed in English, Brazilian Portuguese, Czech, Dutch, French, German, Italian, and Spanish dialog skills because use of the legacy version of the API is no longer supported for those languages. For more information about the new system entities, see [System entities](#).

## 28 October 2020

### Introducing the *actions* skill!

The actions skill is the latest step in the continuing evolution of Watson Assistant as a software as a service application. The actions skill is designed to make it simple enough for *anyone* to build a virtual assistant. We've removed the need to navigate between intents, entities, and dialog to create conversational flows. Building can all now be done in one simple and intuitive interface.

The actions skill is available as a beta feature. For more information, see [Adding an actions skill](#).

### Web chat integration is created automatically

When you create a new assistant, a web chat integration is created for you automatically (in addition to the preview link integration, which was created previously). These integrations are added also to the assistant that is auto-generated (named *My first assistant*) when you create a new service instance. For more information, see [Integrating the web chat with your website](#).

### Text messaging integration was renamed

The *Twilio messaging* integration was renamed to *SMS with Twilio*.

## 9 October 2020

### Search skill update

Support was added for a new version of the Discovery API which adds the following capabilities:

- The search skill can now connect to existing Premium Discovery service instances.
- When you connect to a Box, Sharepoint, or Web crawl data collection, the result content fields are automatically populated for you. The **Title** now uses the `title` field from the source document instead of the `extracted_metadata.title` field, which provides better results.

## 1 October 2020

### Introducing the *Phone* integration!

Your customers are calling; now your assistant can answer. Add a phone integration to enable your assistant to answer customer support calls. The integration connects to your existing Session Initiation Protocol (SIP) trunk, which routes incoming calls to your assistant. For more information, see [Integrating with phone](#).

### Introducing the *Twilio messaging* integration!

Enable your assistant to receive and respond to questions that customers submit by using SMS text messaging. When you enable both new integrations, your assistant can send text messages to a customer in the context of an ongoing phone conversation. For more information, see [Integrating with Twilio messaging](#).

The *Phone* and *Twilio messaging* integrations are available as beta features in Watson Assistant service instances that are hosted in Dallas, Frankfurt, and Washington, DC.

### The web chat integration is added to new assistants automatically

Much like the *Preview link* integration, the *Web chat* integration now is added to the *My first assistant* assistant that is created for new users automatically.

## 24 September 2020

### Introducing the containment metric!

Want a quick way to see how often your assistant has to ask for help? Enable the new containment metric to find out. The containment metric measures the rate at which your assistant is able to address a customer's goal without human intervention. For conversations that are not contained, you can review the logs to understand what led customers to seek help outside of the assistant. For the metric to work, you must design your dialog to flag requests for additional support when they occur. For more information, see [Graphs and statistics](#).

### Chat transfer improvements

When you add the *Connect to human agent* response type to a dialog node, you can now define messages to show to your customers during the transfer, and can specify service desk agent routing preferences. For more information, see [Adding a \*Connect to human agent\* response type](#).

## 22 September 2020

### New API version

The current v2 API version is now `2020-09-24`. In this version, the structure of the `search` response type has changed. The `results` property has been removed and replaced with two new properties:

- `primary_results` property includes the search results that should be displayed in the initial response to a user query.
- `additional_results` property includes search results that can be displayed if the user wants to see more.

The search skill configuration determines how many search results are included in the `primary_results` and `additional_results` properties.

### Search skill improvements

The following improvements were made to the search skill:

- **Control the number of search results:** You can now customize the number of search results that are shown in a response from the search skill. For more information, see [Configure the search](#).
- **FAQ extraction is available for web crawl data collections** When you create a web crawl data collection type, you can now enable the FAQ extraction beta feature. FAQ extraction allows the Discovery service to identify question and answer pairs that it finds as it crawls the website. For more information, see [Create a data collection](#).

## 16 September 2020

### Search skill refinement change

The search refinement beta feature that was added in [June](#) now is disabled by default. Enable the feature to refine the search results that are returned from the Discovery service. For more information, see [Configure the search](#).

## 25 August 2020

### Give the web chat integration a try!

You can now use the web chat integration with a Lite plan. Previously, the web chat was available to Plus or higher plans only. For more information, see [Integrating the web chat with your website](#).

## 12 August 2020

### v2 Logs API is available

If you have a Premium plan, you can use the v2 API `Logs` method to list log events for an assistant. For more information,

see the [API reference](#) documentation.

## 5 August 2020

### Enable your skill to improve itself

Try the new **autolearning** beta feature to empower your skill to improve itself automatically over time. Your skill observes customer choices to understand which choices are most often the best. As its confidence grows, your skill presents better options to get the right answers to your customers with fewer clicks. For more information, see [Empower your skill to learn over time](#).

### Show more of search results

When search results are returned from the search skill, the customer can now click a twistie to expand the search result card to see more of the returned text.

## 29 July 2020

### The @sys-location and @sys-person system entities were removed

The `@sys-location` and `@sys-person` system entities are no longer listed on the *System entities* page. If your dialog uses one of these entities, a red `Entity not created` notification is displayed to inform you that the entity is not recognized.

### Skill menu actions moved

The menu that was displayed in the header of the skill while you were working with a skill was removed. The actions that were available from the menu, such as import and export, are still available. Go to the Skills page, and click the menu on the skill tile.

The import skill process was updated to support overwriting an existing skill on import. For more information, see [Overwriting a skill](#).

### Dialog issues were addressed

These dialog issues were addressed:

- Fixed an issue with adding a jump-to from a conditional response in one node to a conditional response in another node.
- The page now responds better when you scroll horizontally to see multiple levels of child nodes.

## 15 July 2020

### Support ended for @sys-location and @sys-person

The person and location system entities, which were available as a beta feature in English dialog skills only, are no longer supported. You cannot enable them. If your dialog uses them, they are ignored by the service.

Use contextual entities to teach your skill to recognize the context in which such names are used. For more information about contextual entities, see [Annotation-based method](#).

For more information about how to use contextual entities to identify names of people, see the [Detecting Names And Locations With Watson Assistant](#) blog post on Medium.

### **How legacy numeric system entities are processed has changed**

All new dialog skills use the new system entities automatically.

For existing skills that use legacy numeric system entities, how the entities are processed now differs based on the skill language.

- Arabic, Chinese, Korean, and Japanese dialog skills that use legacy numeric system entities function the same as before.
- If you choose to continue to use the legacy system entities in European-language dialog skills, a new legacy API format is used. The new legacy API format simulates the legacy system entities behavior. In particular, it returns a `metadata` object and does not stop the service from identifying multiple system entities for the same input string. In addition, it returns an `interpretation` object, which was introduced with the new version of system entities. Review the `interpretation` object to see the useful information that is returned by the new version.

Update your skills to use the new system entities from the [Options>System Entities](#) page.

### **Web chat security is generally available**

Enable the security feature of web chat so that you can verify that messages sent to your assistant come from only your customers and can pass sensitive information to your assistant.

When configuring the JWT, you no longer need to specify the Authentication Context Class Reference (acr) claim.

## **1 July 2020**

### **Salesforce support is generally available**

Integrate your web chat with Salesforce so your assistant can transfer customers who asks to speak to a person to a Salesforce agent who can answer their questions. For more information, see [Integrating with Salesforce](#).

## **24 June 2020**

### **Get better answers from search skill**

The search skill now has a beta feature that limits the search results that are returned to include only those for which Discovery has calculated a 20% or higher confidence score. You can toggle the feature on or off from the *Refine results to return more selective answers* switch on the configuration page. You cannot change the confidence score threshold from 0.2. This beta feature is enabled by default. For more information, see [Creating a search skill](#).

## **3 June 2020**

### **Zendesk support is generally available**

Integrate your web chat with Zendesk so your assistant can transfer customers who asks to speak to a person to a Zendesk agent who can answer their questions. And now you can secure the connection to Zendesk. For more information, see

[Adding support for transfers.](#)

### Pricing plan changes

We continue to revamp the overall service plan structure for Watson Assistant. In April, we announced [a new low cost entry point](#) for the Plus plan. Today, the Standard plan is being retired. Existing Standard plan users are not impacted; they can continue to work in their Standard instances. New users do not see the Standard plan as an option when they create a service instance. For more information, see the [Pricing](#) page.

## 27 May 2020

### Full language support for new system entities

The new version of the system entities is generally available in dialog skills of all languages, including Arabic, Chinese (Simplified), Chinese (Traditional), Korean, and Japanese. For more information, see [Supported languages](#).

### New system entities are enabled automatically

All new dialog skills use the new version of the system entities automatically. For more information, see [New system entities](#).

## 22 May 2020

### Spelling correction in v2 API

The v2 message API now supports spelling correction options. For more information see the [API Reference](#).

## 21 May 2020

### Preview link URL change

The URL for the preview link was changed. If you previously shared the link with teammates, provide them with the new URL.

## 15 May 2020

### Private endpoints support is available in Plus plan

You can use private endpoints to route services over the IBM Cloud private network instead of the public network. For more information, see [Private network endpoints](#). This feature was previously available to users of Premium plans only.

## 14 May 2020

### Get skill owner information

The email address of the person who owns the service instance that you are using is displayed from the User account menu. This information is especially helpful if you want to contact the instance owner to request access changes. For more information about access control, see [Managing access to resources](#).

## System entity depreciation

As stated in the [March deprecation notice](#), the @sys-location and @sys-person system entities that were available as a beta feature are deprecated. If you are using one of these system entities in your dialog, a toggle is displayed for the entity on the *System entities* page. You can [search your dialog](#) to find out where you are currently using the entity, and remove it. Consider using a contextual entity to identify references to locations and people instead. After removing the entity from your dialog, disable the entity from the *System entities* page.

## 13 May 2020

### Stateless v2 message API

The v2 runtime API now supports a new statelessmessage method. If you have a client application that manages its own state, you can use this new method to take advantage of [many of the benefits](#) of the v2 API without the overhead of creating sessions. For more information, see the [API Reference](#).

## 30 April 2020

### Web chat is generally available!

Add your assistant to your company website as a web chat widget that can help your customers with common questions and tasks. Service desk transfer support continues to be a beta feature. For more information, see [Integrating with your own website](#).

### Secure your web chat

Enable the beta security feature of web chat so that you can verify that messages sent to your assistant come from only your customers and can pass sensitive information to your assistant.

## 27 April 2020

### Add personality to your assistant in web chat

You can add an assistant image to the web chat header to brand the window. You can add an avatar image that represents your assistant or a brand logo, for example. For more information, see [Integrating with your own web site](#).

### Know your plan

Now your service plan is displayed in the page header. And if you have a Plus Trial plan, you can see how many days are left in the trial.

## 21 April 2020

### Fuzzy matching support was expanded

Added support for stemming and misspelling in French, German, and Czech dialog skills. This enhancement means that the assistant can recognize an entity value that is defined in its singular form but mentioned in its plural form in user input. It also can recognize conjugated forms of a verb that is specified as an entity value.

For example, if your French-language dialog skill has an entity value of `animal`, it recognizes the plural form of the word (`animaux`) when it is mentioned in user input. If your German-language dialog skill has the root verb `haben` as an entity value, it recognizes conjugated forms of the verb (`hast`) in user input as mentions of the entity.

## 2 April 2020

### New and improved access control

Now, when you give other people access to your Watson Assistant resources, you have more control over the level of access they have to individual skills and assistants. You can give one person read-only access to a production skill and manager-level access to a development skill, for example. For more information, see [Managing access to resources](#).

Can't see Analytics anymore? If you cannot do things that you could do before, you might not have appropriate access. Ask the service instance owner to change your service access role. For more information, see [How to keep your access](#).

If you can't access the API Details for a skill or assistant anymore, you might not have the access role that is required to use the instance-level API credentials. You can use a personal API key instead. For more information, see [Getting API information](#).

## 1 April 2020

### Plus plan changes

The Plus plan is now available starting at \$120/month for 1,000 users on pay-as-you-go or subscription IBM Cloud accounts. And you can subscribe without contacting Sales.

### French language beta support added for contextual entities

You can add contextual entities to French-language dialog skills. For more information about contextual entities, see [Creating entities](#).

### New API version

The current API version is now `2020-04-01`. The following change was made with this version:

- An `integrations` property was added to the V2 `/message` context. The service now expects the `context.integrations` property to conform to a specific schema in which the allowed values are as follows:
  - `chat`
  - `facebook`
  - `intercom`
  - `liveengage`
  - `salesforce`
  - `slack`
  - `service_desk`
  - `text.messaging`
  - `voice.telephony`

- zendesk

If your app uses a `context.integrations` property that does not conform to the schema, a 400 error code will be returned.

## 31 March 2020

### **The web chat integration was updated**

The update adds an `isTrackingEnabled` parameter. You can add this parameter and set it to `false` to add the `X-Watson-Learning-Opt-Out` header to each `/message` request that originates from the web chat. For more information about the header, see [Data collection](#). For more information about the parameter, see [Configuration](#).

## 26 March 2020

### **The Covid-19 content catalog is available in Brazilian Portuguese, French, and Spanish**

The content catalog defines a group of intents that recognize the common types of questions people ask about the novel coronavirus. You can use the catalog to jump-start development of chatbots that can answer questions about the virus and help to minimize the anxiety and misinformation associated with it. For more information about how to add a content catalog to your skill, see [Using content catalogs](#).

## 19 March 2020

### **A Covid-19 content catalog is available**

The English-only content catalog defines a group of intents that recognize the common types of questions people ask about the novel coronavirus. The World Health Organization characterized COVID-19 as a pandemic on 11 March 2020. You can use the catalog to jump-start development of chatbots that can answer questions about the virus and help to minimize the anxiety and misinformation associated with it. For more information about how to add a content catalog to your skill, see [Using content catalogs](#).

### **Fixed a problem with missing User Conversation data**

A recent change resulted in no logs being shown in the User Conversations page unless you had a skill as the chosen data source. And the chosen skill had to be the same skill (with same skill ID) that was connected to the assistant when the user messages were submitted.

## 18 March 2020

### **Technology preview is discontinued**

The technology preview user interface was replaced with the Watson Assistant standard user interface. If you used an Actions page to create actions and steps for your skill previously, you cannot access the Actions page anymore. Instead, use the Intents and Dialog pages to work with your skill.

## 16 March 2020

### Instructions updated for Slack integrations

The steps required to set up a Slack integration have changed to reflect permission assignment changes that were made by Slack. For more information, see [Integrating with Slack](#).

### Order of response types is preserved

Previously, if you included a response type of **Search skill** in a list of response types for a dialog node, the search results were displayed last despite its placement in the list. This behavior was changed to show the search results in the appropriate order, namely in the sequence in which the search skill response type is listed for the dialog node.

## 10 March 2020

### Contextual entity support is generally available

You can add contextual entities to English-language dialog skills. For more information about contextual entities, see [Creating entities](#).

### French language support added for autocorrection

Autocorrection helps your assistant understand what your customers want. It corrects misspellings in the input that customers submit before the input is evaluated. With more precise input, your assistant can more easily recognize entity mentions and understand the customer's intent. See [Correcting user input](#) for more details.

### The new system entities are used by new skills

For new English, Brazilian Portuguese, Czech, Dutch, French, German, Italian, and Spanish dialog skills, the new system entities are enabled automatically. If you decide to turn on a system entity and add it to your dialog, it's the new and improved version of the system entity that is used. For more information, see [New system entities](#).

## 6 March 2020

### Transfer a web chat conversation to a human agent

Delight your customers with 360-degree support by integrating your web chat with a third-party service desk solution. When a customer asks to speak to a person, you can connect them to an agent through a service desk solution, such as Zendesk or Salesforce. Service desk support is a beta feature. For more information, see [Adding support for transfers](#).

## 2 March 2020

### Known issue accessing logs

If you cannot access user logs from the Analytics page, ask the owner of the service instance for the skill to change your service level access to make you a Manager of the instance. For more information about access control, see [Managing access to resources](#).

## 1 March 2020 deprecation notice

## March 2020 deprecation notice

To help us continue to improve and expand the capabilities of the assistants you build with Watson Assistant, we are deprecating some of the older technologies. Support for the older technologies will end in June 2020. Take action now to test and adopt the new technologies, so your skills and assistants will be ready when the old technologies stop being supported.

The following technologies are being deprecated:

- **Legacy version of numeric system entities**

We released a whole new infrastructure for our numeric system entities across all languages except Chinese, Korean, Japanese and Arabic. The updated `@sys-number`, `@sys-date`, `@sys-time`, `@sys-currency`, and `@sys-percentage` entities provide superior number recognition with higher precision. For more information about the new system entities, see [System entity details](#).

The old version of the numeric system entities will stop being supported in June 2020 for English, Brazilian Portuguese, Czech, Dutch, French, German, Italian, and Spanish dialog skills.

**Action:** In each dialog skill where you use numeric system entities, go to the [Options>System entities](#) page and turn on the new system entities. Take some time to test the new version of system entities with your own dialogs to make sure they continue to work as expected. As you adopt the new system entities, share your feedback about your experience with the new technology.

- **Person and location system entities**

The `@sys-person` and `@sys-location` system entities, which were available in English as a beta only, are being deprecated. Consider using contextual entities as a way to capture these types of proper nouns. Instead of trying to add a dictionary-based entity that covers every permutation of the names for people or cities, for example, you can teach your skill to recognize the context in which such names are used. For more information about contextual entities, see [Annotation-based method](#).

**Action:** Remove references to `@sys-person` and `@sys-location` from your dialogs. Turn off the `@sys-person` and `@sys-location` system entities to prevent yourself or others from adding them to a dialog inadvertently.

- **Irrelevance detection**

We revised the irrelevance detection classification algorithm to make it even smarter out of the box. Now, even before you begin to teach the system about irrelevant requests, it is able to recognize user input that your skill is not designed to address. For more information, see [Irrelevance detection](#).

**Action:** In each dialog skill, go to the [Options>Irrelevance detection](#) page and turn on the new classification model. Make sure everything works as well, if not better, than it did before. Share your feedback.

- **Old API version dates**

v1 API versions that are dated on or before `2017-02-03` are being deprecated. When you send calls to the service with earlier API version dates, they will receive properly formatted and valid responses for a time, so you can gracefully transition to using the later API versions. However, the confidence scores and other results that are sent in the response will reflect those generated by a more recent version of the API.

**Action:** Do some testing of calls with the latest version to verify that things work as expected. Some functionality has changed over the last few years. After testing, change the version date on any API calls that you make from your applications.

## 28 February 2020

### IBM Watson® Assistant is available in IBM Cloud Pak for Data

The service can be installed on-premises in environments where IBM Cloud Pak for Data 2.5 is installed on OpenShift or standalone. See the [IBM Cloud Pak for Data documentation](#) for more information.

## 26 February 2020

### Slot Save it as field retains your edits

When you edit what gets saved for a slot by using the JSON editor to edit the value of the context variable to be something other than what is specified in the **Check for** field, your changes are kept even if someone subsequently clicks the **Save it as** field.

## 20 February 2020

### Access control changes are coming

Notifications are displayed in the user interface for anyone with Reader and Writer level access to a service instance. The notification explains that access control is going to change soon, and that what they can do in the instance will change unless they are given Manager service access beforehand. For more information, see [Preventing loss of access](#).

## 14 February 2020

### More web chat color settings

You can now specify the color of more elements of the web chat integration. For example, you can define one color for the web chat window header. You can define a different color for the user message bubble. And another color for interactive components, such as the launcher button for the chat.

## 13 February 2020

### Track API events

Premium plan users can now use the Activity Tracker service to track how users and applications interact with IBM Watson® Assistant in IBM Cloud®. See [Activity Tracker events](#).

## 5 February 2020

### New API version

The current API version is now `2020-02-05`. The following changes were made with this version:

- When a dialog node's response type is `connect-to-agent`, the node's `title` is used as the `topic` value. Previously, `user_label` was used.
- The `alternate_intents` property is stored as a Boolean value instead of a String.

## 4 February 2020

### **Product user interface makeover**

The UI has been updated to be more intuitive, responsive, and consistent across its pages. While the look and feel of the UI elements has changed, their function has not.

### **Requesting early access**

The button you click to request participation in the early access program has moved from the Skills page to the user account menu. For more information, see [Feedback](#).

## 24 January 2020

### **New system entities are now generally available in multiple languages**

The new and improved numeric system entities are now generally available in all supported languages, except Arabic, Chinese, Japanese, and Korean, where they are available as a beta feature. They are not used by your dialog skill unless you enable them from the **Options>System entities** page. For more information, see [New system entities](#).

## 14 January 2020

### **Fixed an error message that was displayed when opening an instance**

An error that was displayed when you launched Watson Assistant from the IBM Cloud® dashboard has been fixed. Previously, an error message that said, `Module 'ui-router' is not available! You either misspelled the module name or forgot to load it` would sometimes be displayed.

## 12 December 2019

### **Support for private network endpoints**

Users of Premium plans can create private network endpoints to connect to Watson Assistant over a private network. Connections to private network endpoints do not require public internet access. For more information, see [Protecting sensitive information](#).

### **Full support for IBM Cloud IAM**

Watson Assistant now supports the full implementation of IBM Cloud Identity and Access Management (IAM). API keys for Watson services are no longer limited to a single service instance. You can create access policies and API keys that apply to more than one service, and you can grant access between services.

- To support this change, the API service endpoints use a different domain and include the service instance ID. The pattern is `api.{location}.{offering}.watson.cloud.ibm.com/instances/{instance_id}`.

Example URL for an instance hosted in the Dallas location: `api.us-south.assistant.watson.cloud.ibm.com/instances/6bbda3b3-d572-45e1-8c54-22d6ed9e52c2`

The previous public endpoint domain was `watsonplatform.net`.

For more information, see the [API reference](#).

These URLs do not introduce a breaking change. The new URLs work both for your existing service instances and for new instances. The original URLs continue to work on your existing service instances for at least one year (until December 2020).

- For more information, see [Authenticating to Watson services](#).

## 26 November 2019

### Disambiguation is available to everyone

Disambiguation is now available to users of every plan type.

The following changes were made to how it functions:

- The text that you add to the dialog **node name** field now matters.
- The text in the node name field might be shown to customers. The disambiguation feature shows it to customers if the assistant needs to ask them to clarify their meaning. The text you add as the node name must identify the purpose of the node clearly and succinctly, such as *Place an order* or *Get plan information*.

If the *External node name* field exists and contains a summary of the node's purpose, then its summary is shown in the disambiguation list instead. Otherwise, the dialog node name content is shown.

- Disambiguation is enabled automatically for all nodes. You can disable it for the entire dialog or for individual dialog nodes.
- When testing, you might notice that the order of the options in the disambiguation list changes from one test run to the next. Don't worry; this new behavior is intended. As part of work being done to help the assistant learn automatically from user choices, the order of the options in the disambiguation list is being randomized on purpose. Changing the order helps to avoid bias that can be introduced by a percentage of people who always pick the first option without first reviewing their choices.

## 12 November 2019

### Slot prompt JSON editor

You can now use the context or JSON editors for the slot response field where you define the question that your assistant asks to get information it needs from the customer. For more information about slots, see [Gathering information with slots](#).

### New South Korea location

You can now create Watson Assistant instances in the Seoul location. As with other locations, the IBM Cloud Seoul location

uses token-based Identity and Access Management (IAM) authentication.

## Technology preview

A technology preview experience was released. A select set of new users are being presented with a new user interface that takes a different approach to building an assistant.

## 7 November 2019

### Irrelevance detection has been added

When enabled, a supplemental model is used to help identify utterances that are irrelevant and should not be answered by the dialog skill. This new model is especially beneficial for skills that have not been trained on what subjects to ignore. This feature is available for English skills only. For more information, see [Irrelevance detection](#).

### Time zone support for now() method

You can now specify the time zone for the date and time that is returned by the `now()` method. See [Now\(\)](#).

## 24 October 2019

### Testing improvement

You can now see the top three intents that were recognized in a test user input from the "Try it out" pane. For more details, see [Testing your dialog](#).

### Error message when opening an instance

When you launch Watson Assistant from the IBM Cloud® dashboard, you might see an error message that says, `Module 'ui-router' is not available! You either misspelled the module name or forgot to load it.` You can ignore the message. Refresh the web browser page to close the notification.

## 16 October 2019

The changes from 14 October are now available in Dallas.

## 14 October 2019

### Deploy your assistant in minutes

Create a web chat integration to embed your assistant into a page on your website as a chat widget. See [Integrating with your own website](#).

### UI changes

The main menu options of **Assistants** and **Skills** have moved from being displayed in the page header to being shown as icons on the side of the page. The tabbed pages for the tools you use to develop a dialog skill were moved to a secondary navigation bar that is displayed when you open the skill.

### Rich response types are supported in a dialog node with slots

You can display a list of options for a user to choose from as the prompt for a slot, for example.

### Change to switching service instances

Where you go to switch service instances has changed. See [Switching the service instance](#).

#### **Known issue: Cannot rename search skills**

You currently cannot rename a search skill after you create it.

## **9 October 2019**

#### **New system entities changes**

The following updates have been made:

- In addition to English and German, the new numeric system entities are now available in these languages: Brazilian Portuguese, Czech, French, Italian, and Spanish.
- The `part_of_day` property of the `@sys-time` entity now returns a time range instead of a single time value.

## **23 September 2019**

#### **Dallas updates**

The updates from 20 September are now available to service instances hosted in Dallas.

## **20 September 2019**

#### **Inactivity timeout increase**

The maximum inactivity timeout can now be extended to up to 7 days for Premium plans. See [Changing the inactivity timeout setting](#).

#### **Pattern entity fix**

A change that was introduced in the previous release which changed all alphabetic characters to lowercase at the time an entity value was added has been fixed. The case of any alphabetic characters that are part of a pattern entity value are no longer changed when the value is added.

#### **Dialog text response syntax fix**

Fixed a bug in which the format of a dialog response reverted to an earlier version of the JSON syntax. Standard text responses were being saved as `output.text` instead of `output.generic`. For more information about the `output` object, see [Anatomy of a dialog call](#).

## **13 September 2019**

#### **Improved Entities and Intents page responsiveness**

The Entities and Intents pages were updated to use a new JavaScript library that increases the page responsiveness. As a result, the look of some graphical user interface elements, such as buttons, changed slightly, but the function did not.

#### **Creating contextual entities got easier**

The process you use to annotate entity mentions from intent user examples was improved. You can now put the intent page into annotation mode to more easily select and label mentions. See [Adding contextual entities](#).

## 6 September 2019

### **Label character limit increase**

The limit to the number of characters allowed for a label that you define for an option response type changed from 64 characters to 2,048 characters.

## 12 August 2019

### **New dialog method**

The `getMatch` method was added. You can use this method to extract a specific occurrence of a regular expression pattern that recurs in user input. For more details, see the [dialog methods](#) topic.

## 9 August 2019

### **Introductory product tour**

For some first-time users, a new introductory product tour is shown that the user can choose to follow to perform the initial steps of creating an assistant.

## 6 August 2019

- Webhook callouts and Dialog page improvements are available in Dallas.

## 1 August 2019

### **Webhook callouts are available**

Add webhooks to dialog nodes to make programmatic calls to an external application as part of the conversational flow. The new Webhook support simplifies the callout implementation process. (No more `action` JSON objects required.) For more information, see [Making a programmatic call from a dialog node](#).

### **Improved dialog page responsiveness**

In all service instances, the user interface of the Dialog page was updated to use a new JavaScript library that increases the page responsiveness. As a result, the look of some graphical user interface elements, such as buttons, changed slightly, but the function did not.

## 31 July 2019

### **Search skill and autocorrection are generally available**

The search skill and spelling autocorrection features, which were previously available as beta features, are now generally available.

- Search skills can be created by users of Plus or Premium plans only.
- You can enable autocorrection for English-language dialog skills only. It is enabled automatically for new English-language dialog skills.

## 26 July 2019

### **Missing skills issue is resolved**

In some cases, workspaces that were created through the API only were not being displayed when you opened the Watson Assistant user interface. This issue has been addressed. All workspaces that you create by using the API are displayed as dialog skills when you open the user interface.

## 23 July 2019

### **Dialog search is fixed**

In some skills, the search function was not working in the Dialog page. The issue is now fixed.

## 17 July 2019

### **Disambiguation choice limit**

You can now set the maximum number of options to show to users when the assistant asks them to clarify what they want to do. For more information about disambiguation, see [Disambiguation](#).

### **Dialog search issue**

In some skills, the search function is not working in the Dialog page. A new user interface library, which increases the page responsiveness, is being rolled out to existing service instances in phases. This search issue affects only dialog skills for which the new library is not yet enabled.

### **Missing skills issue**

In some cases, workspaces that were created through the API only are not being displayed when you open the Watson Assistant user interface. Normally, these workspaces are displayed as dialog skills. If you do not see your skills from the UI, don't worry; they are not gone. Contact support to report the issue, so the team can enable the workspaces to be displayed properly.

## 15 July 2019

### **Numeric system entities upgrade available in Dallas**

The new system entities are now also available as a beta feature for instances that are hosted in Dallas. See [New system entities](#).

## 12 June 2019

### Numeric system entities upgrade

New system entities are available as a beta feature that you can enable in dialog skills that are written in English or German. The revised system entities offer better date and time understanding. They can recognize date and number spans, national holiday references, and classify mentions with more precision. For example, a date such as May 15 is recognized as a date mention(@sys-date:2019-05-15), and is *not* also identified as a number mention (@sys-number:15). See [New system entities](#).

### A Plus Trial plan is available

You can use the free Plus Trial plan to try out the features of the Plus plan as you make a purchasing decision. The trial lasts for 30 days. After the trial period ends, if you do not upgrade to a Plus plan, your Plus Trial instance is converted to a Lite plan instance.

## 23 May 2019

### Updated navigation

The home page was removed, and the order of the Assistants and Skills tabs was reversed. The new tab order encourages you to start your development work by creating an assistant, and then a skill.

### Disambiguation settings have moved

The toggle to enable disambiguation, which is a feature that is available to Plus and Premium plan users only, has moved. The **Settings** button was removed from the **Dialog** page. You can now enable disambiguation and configure it from the skill's **Options** tab.

### An introductory tour is now available

A short product tour is now displayed when a new service instance is created. Brand new users are also given help as they start development. A new assistant is created for them automatically. Informational popups are displayed to introduce the product user interface features, and guide the new user toward taking the key first step of creating a dialog skill.

## 10 April 2019

### Autocorrection is now available

Autocorrection is a beta feature that helps your assistant understand what your customers want. It corrects misspellings in the input that customers submit before the input is evaluated. With more precise input, your assistant can more easily recognize entity mentions and understand the customer's intent. See [Correcting user input](#) for more details.

## 22 March 2019

### Introducing search skill

A search skill helps you to make your assistant useful to customers faster. Customer inquiries that you did not anticipate and so have not built dialog logic to handle can be met with useful responses. Instead of saying it can't help, the assistant can

query an external data source to find relevant information to share in its response. Over time, you can build dialog responses to answer customer queries that require follow-up questions to clarify the user's meaning or for which a short and clear response is suitable. And you can use search skill responses to address more open-ended customer queries that require a longer explanation. This beta feature is available to users of Premium and Plus service plans only.

See [Building a search skill](#) for more details.

## 14 March 2019

## 4 March 2019

### Simplified navigation

The sidebar navigation with separate *Build*, *Improve*, and *Deploy* tabs has been removed. Now, you can get to all the tools you need to build a dialog skill from the main skill page.

### Improve page is now called Analytics

The informational metrics that Watson generates from conversations between your users and your assistant moved from the *Improve* tab of the sidebar to a new tab on the main skill page called **Analytics**.

## 1 March 2019

## 28 February 2019

### New API version

The current API version is now `2019-02-28`. The following changes were made with this version:

- The order in which conditions are evaluated in nodes with slots has changed. Previously, if you had a node with slots that allowed for digressions away, the `anything_else` root node was triggered before any of the slot level Not found conditions could be evaluated. The order of operations has been changed to address this behavior. Now, when a user digresses away from a node with slots, all the root nodes except the `anything_else` node are processed. Next, the slot level Not found conditions are evaluated. And, finally, the root level `anything_else` node is processed. To better understand the full order of operations for a node with slots, see [Slot usage tips](#).
- Strings that begin with a number sign (#) in the `context` or `output` objects of a message are no longer treated as intent references.

Previously, these strings were treated as intents automatically. For example, if you specified a context variable, such as `"color" : "#FFFFFF"`, then the hex color code (#FFFFFF) would be treated as an intent. Your assistant would check whether an intent named #FFFFFF was detected in the user's input, and if not, would replace #FFFFFF with `false`. This replacement no longer occurs.

Similarly, if you included a number sign (#) in the text string in a node response, you used to have to escape it by preceding it with a back slash (\). For example, `We are the \#1 seller of lobster rolls in Maine.` You no longer need to escape the # symbol in a text response.

This change does not apply to node or conditional response conditions. Any strings that begin with a number sign (#) which are specified in conditions continue to be treated as intent references. Also, you can use SpEL expression syntax

to force the system to treat a string in the `context` or `output` objects of a message as an intent. For example, specify the intent as `<? #intent-name ?>`.

## 25 February 2019

### Slack integration enhancement

You can now choose the type of event that triggers your assistant in a Slack channel. Previously, when you integrated your assistant with Slack, the assistant interacted with users through a direct message channel. Now, you can configure the assistant to listen for mentions, and respond when it is mentioned in other channels. You can choose to use one or both event types as the mechanism through which your assistant interacts with users.

## 11 February 2019

### Integrate with Intercom

Intercom, a leading customer service messaging platform, has partnered with IBM to add a new agent to the team, a virtual Watson Assistant. You can integrate your assistant with an Intercom application to enable the app to seamlessly pass user conversations between your assistant and human support agents. This integration is available to Plus and Premium plan users only. See [Integrating with Intercom](#) for more details.

## 8 February 2019

### Version your skills

You can now capture a snapshot of the the intents, entities, dialog, and configuration settings for a skill at key points during the development process. With versions, it's safe to get creative. You can deploy new design approaches in a test environment to validate them before you apply any updates to a production deployment of your assistant. See [Creating skill versions](#) for more details.

### Arabic content catalog

Users of Arabic-language skills can now add prebuilt intents to their dialogs. See [Using content catalogs](#) for more information.

## 17 January 2019

### Czech language support is generally available

Support for the Czech language is no longer classified as beta; it is now generally available. See [Supported languages](#) for more information.

### Language support improvements

The language understanding components were updated to improve the following features:

- German and Korean system entities

- Intent classification tokenization for Arabic, Dutch, French, Italian, Japanese, Portuguese, and Spanish

## 4 January 2019

### IBM Cloud Functions in DC and London locations

You can now make programmatic calls to IBM Cloud Functions from the dialog of an assistant in a service instance that is hosted in the London and Washington, DC data centers. See [Making programmatic calls from a dialog node](#).

### New methods for working with arrays

The following SpEL expression methods are available that make it easier to work with array values in your dialog:

- JSONArray.filter**: Filters an array by comparing each value in the array to a value that can vary based on user input.
- JSONArray.includesIntent**: Checks whether an `intents` array contains a particular intent.
- JSONArray.indexOf**: Gets the index number of a specific value in an array.
- JSONArray.joinToArray**: Applies formatting to values that are returned from an array.

See the [array method documentation](#) for more details.

## 13 December 2018

### London data center

You can now create Watson Assistant service instances that are hosted in the London data center without syndication. See [Data centers](#) for more details.

### Dialog node limit changes

The dialog node limit was temporarily changed from 100,000 to 500 for new Standard plan instances. This limit change was later reversed. If you created a Standard plan instance during the time frame in which the limit was in effect, your dialogs might be impacted. The limit was in effect for skills created between 10 December and 12 December 2018. The lower limits will be removed from all impacted instances in January. If you need to have the lower limit lifted before then, open a support ticket.

## 1 December 2018

### Determine the number of dialog nodes

To determine the number of dialog nodes in a dialog skill, do one of the following things:

- From the tool, if it is not associated with an assistant already, add the dialog skill to an assistant, and then view the skill tile from the main page of the assistant. The *trained data* section lists the number of dialog nodes.
- Send a GET request to the `/dialog_nodes` API endpoint, and include the `include_count=true` parameter. For example:

```
curl -u "apikey:{apikey}" "https://{{service-
hostname}}/assistant/api/v1/worksaces/{{workspace_id}}/dialog_nodes?version=2018-09-
```

```
20&include_count=true"
```

where {service-hostname} is the appropriate URL for your instance. For more details, see [Service endpoint](#).

In the response, the `total` attribute in the `pagination` object contains the number of dialog nodes.

See [Troubleshooting skill import issues](#) for information about how to edit skills that you want to continue using.

## 27 November 2018

### A new service plan, the Plus plan, is available

The new plan offers premium-level features at a lower price point. Unlike previous plans, the Plus plan is a user-based billing plan. It measures usage by the number of unique users that interact with your assistant over a given time period. To get the most from the plan, if you build your own client application, design your app such that it defines a unique ID for each user, and passes the user ID with each /message API call. For the built-in integrations, the session ID is used to identify user interactions with the assistant. See [User-based plans](#) for more information.

Artifact	Limit
Assistants	100
Contextual entities	20
Contextual entity annotations	2,000
Dialog nodes	100,000
Entities	1,000
Entity synonyms	100,000
Entity values	100,000
Intents	2,000
Intent user examples	25,000
Integrations	100
Logs	30 days
Skills	50

#### Plus plan limits

### User-based Premium plan

The Premium plan now bases its billing on the number of active unique users. If you choose to use this plan, design any custom applications that you build to properly identify the users who generate /message API calls. See [User-based plans](#) for more information.

Existing Premium plan service instances are not impacted by this change; they continue to use API-based billing methods. Only existing Premium plan users will see the API-based plan listed as the *Premium (API)* plan option.

See Watson Assistant [service plan options](#) for more information about all available service plans.

## 20 November 2018

### **\*\*Recommendations are discontinued**

The Recommendations section on the Improve tab was removed. Recommendations was a beta feature available to Premium plan users only. It recommended actions that users could take to improve their training data. Instead of consolidating recommendations in one place, recommendations are now being made available from the parts of the tool where you make actual training data changes. For example, while adding entity synonyms, you can now opt to see a list of synonymous terms that are recommended by Watson. If you are looking for other ways to analyze your user conversation logs in more detail, consider using Jupyter notebooks. See [Advanced tasks](#) for more details.

## 9 November 2018

### **Major user interface revision**

The Watson Assistant service has a new look and added features.

This version of the tool was evaluated by beta program participants over the past several months.

- **Skills:** What you think of as a *workspace* is now called a *skill*. A *dialog skill* is a container for the natural language processing training data and artifacts that enable your assistant to understand user questions, and respond to them.

**Where are my workspaces?** Any workspaces that you created previously are now listed in your service instance as skills. Click the **Skills** tab to see them. For more information, see [Adding skills to your assistant](#).

- **Assistants:** You can now publish your skill in just two steps. Add your skill to an assistant, and then set up one or more integrations with which to deploy your skill. The assistant adds a layer of function to your skill that enables Watson Assistant to orchestrate and manage the flow of information for you. See [Assistants](#).
- **Built-in integrations:** Instead of going to the **Deploy** tab to deploy your workspace, you add your dialog skill to an assistant, and add integrations to the assistant through which the skill is made available to your users. You do not need to build a custom front-end application and manage the conversation state from one call to the next. However, you can still do so if you want to. See [Adding integrations](#) for more information.
- **New major API version:** A V2 version of the API is available. This version provides access to methods you can use to interact with an assistant at run time. No more passing context with each API call; the session state is managed for you as part of the assistant layer.

What is presented in the tooling as a dialog skill is effectively a wrapper for a V1 workspace. There are currently no API methods for authoring skills and assistants with the V2 API. However, you can continue to use the V1 API for authoring workspaces. See [API Overview](#) for more details.

- **Switching data sources:** It is now easier to improve the model in one skill with user conversation logs from a different skill. You do not need to rely on deployment IDs, but can simply pick the name of the assistant to which a skill was added and deployed to use its data. See [Improving across assistants](#).

- **Preview links from London instances:** If your service instance is hosted in London, then you must edit the preview link URL. The URL includes a region code for the region where the instance is hosted. Because instances in London are syndicated to Dallas, you must replace the `eu-gb` reference in the URL with `us-south` for the preview web page to render properly.

## 8 November 2018

### Japanese data center

You can now create Watson Assistant service instances that are hosted in the Tokyo data center. See [Data centers](#) for more details.

## 30 October 2018

### New API authentication process

The Watson Assistant service transitioned from using Cloud Foundry to using token-based Identity and Access Management (IAM) authentication in the following regions:

- Dallas (us-south)
- Frankfurt (eu-de)

For new service instances, you use IAM for authentication. You can pass either a bearer token or an API key. Tokens support authenticated requests without embedding service credentials in every call. API keys use basic authentication.

For all existing service instances, you continue to use service credentials `{username}:{password}` for authentication.

## 25 October 2018

### Entity synonym recommendations are available in more languages

Synonym recommendation support was added for the French, Japanese, and Spanish languages.

## 26 September 2018

### IBM Watson® Assistant is available in IBM® Cloud Private

IBM Watson® Assistant is available in IBM® Cloud Private

## 21 September 2018

### New API version

The current API version is now 2018-09-20. In this version, the `errors[] .path` attribute of the error object that is returned

by the API is expressed as a [JSON Pointer](#) instead of in dot notation form.

## Web actions support

You can now call Cloud Functions web actions from a dialog node. See [Making programmatic calls from a dialog node](#) for more details.

## 15 August 2018

### Entity fuzzy matching support improvements

Fuzzy matching is fully supported for English entities, and the misspelling feature is no longer a Beta-only feature for many other languages. See [Supported languages](#) for details.

## 6 August 2018

### Intent conflict resolution

The tool can now help you to resolve conflicts when two or more user examples in separate intents are similar to one another. Non-distinct user examples can weaken the training data and make it harder for your assistant to map user input to the appropriate intent at run time. See [Resolving intent conflicts](#) for details.

### Disambiguation

Enable disambiguation to allow your assistant to ask the user for help when it needs to decide between two or more viable dialog nodes to process for a response. See [Disambiguation](#) for more details.

### Jump-to fix

Fixed a bug in the Dialogs tool which prevented you from being able to configure a jump-to that targets the response of a node with the `anything_else` special condition.

### Digression return message

You can now specify text to display when the user returns to a node after a digression. The user will have seen the prompt for the node already. You can change the message slightly to let users know they are returning to where they left off. For example, specify a response like, `Where were we? Oh, yes...` See [Digressions](#) for more details.

## 12 July 2018

### Rich response types

You can now add rich responses that include elements such as images or buttons in addition to text, to your dialog. See [Rich responses](#) for more information.

### Contextual entities (Beta)

Contextual entities are entities that you define by labeling mentions of the entity type that occur in intent user examples. These entity types teach your assistant not only terms of interest, but also the context in which terms of interest typically appear in user utterances, enabling your assistant to recognize never-seen-before entity mentions based solely on how they are referenced in user input. For example, if you annotate the intent user example, `I want a flight to Boston` by labeling `Boston` as a `@destination` entity, then your assistant can recognize `Chicago` as a `@destination` mention in a

user input that says, `I want a flight to Chicago.` This feature is currently available for English only. See [Adding contextual entities](#) for more information.

When you access the tool with an Internet Explorer web browser, you cannot label entity mentions in intent user examples nor edit user example text.

## Entity recommendations

Watson can now recommend synonyms for your entity values. The recommender finds related synonyms based on contextual similarity extracted from a vast body of existing information, including large sources of written text, and uses natural language processing techniques to identify words similar to the existing synonyms in your entity value. For more information see [Synonyms](#).

## New API version

The current API version is now `2018-07-10`. This version introduces the following changes:

- The content of the `/message output` object changed from being a `text` JSON object to being a `generic` array that supports multiple rich response types, including `image`, `option`, `pause`, and `text`.
- Support for contextual entities was added.
- You can no longer add user-defined properties in `context.metadata`. However, you can add them directly to `context`.

## Overview page date filter

Use the new date filters to choose the period for which data is displayed. These filters affect all data shown on the page: not just the number of conversations displayed in the graph, but also the statistics displayed along with the graph, and the lists of top intents and entities. See [Controls](#) for more information.

## Pattern limit expanded

When using the **Patterns** field to [define specific patterns for an entity value](#), the pattern (regular expression) is now limited to 512 characters.

## 2 July 2018

### Jump-tos from conditional responses

You can now configure a conditional response to jump directly to another node. See [Conditional responses](#) for more details.

## 21 June 2018

### Language updates for system entities

Dutch and Simplified Chinese language support are now generally available. Dutch language support includes fuzzy matching for misspellings. Traditional Chinese language support includes the availability of `system entities` in beta release. See [Supported languages](#) for details.

## 14 June 2018

## Washington, DC data center opens

You can now create Watson Assistant service instances that are hosted in the Washington, DC data center. See [Data centers](#) for more details.

## New API authentication process

The Watson Assistant service has a new API authentication process for service instances that are hosted in the following regions:

- Washington, DC (us-east) as of 14 June 2018
- Sydney, Australia (au-syd) as of 7 May 2018

IBM Cloud® is migrating to token-based Identity and Access Management (IAM) authentication.

For new service instances in the regions listed, you use IAM for authentication. You can pass either a bearer token or an API key. Tokens support authenticated requests without embedding service credentials in every call. API keys use basic authentication.

For all new and existing service instances in other regions, you continue to use service credentials `{username}: {password}` for authentication.

When you use any of the Watson SDKs, you can pass the API key and let the SDK manage the lifecycle of the tokens. For more information and examples, see [Authentication](#) in the API reference.

If you are not sure which type of authentication to use, view the Watson Assistant credentials by clicking the service instance from the Services section of the [IBM Cloud Resource List](#).

## 25 May 2018

### New sample workspace

The sample workspace that is provided for you to explore or to use as a starting point for your own workspace has changed. The **Car Dashboard** sample was replaced by a **Customer Service** sample. The new sample showcases how to use content catalog intents and other newer features to build a bot. It can answer common questions, such as inquiries about store hours and locations, and illustrates how to use a node with slots to schedule in-store appointments.

### HTML rendering was added to Try it out

The "Try it out" pane now renders HTML formatting that is included in response text. Previously, if you included a hypertext link as an HTML anchor tag in a text response, you would see the HTML source in the "Try it out" pane during testing. It used to look like this:

Contact us at <a href="https://www.ibm.com">ibm.com</a>.

Now, the hypertext link is rendered as if on a web page. It is displayed like this:

Contact us at [ibm.com](https://www.ibm.com).

Remember, you must use the appropriate type of syntax in your responses for the client application to which you will deploy the conversation. Only use HTML syntax if your client application can interpret it properly. Other integration channels might expect other formats.

## Deployment changes

The **Test in Slack** option was removed.

## 11 May 2018

### Information security

The documentation includes some new details about data privacy. Read more in [Information security](#).

## 7 May 2018

### Sydney, Australia data center opens

You can now create Watson Assistant service instances that are hosted in the Sydney, Australia data center. See [IBM Cloud global data centers](#) for more details.

## 4 April 2018

### Search dialogs

You can now [search dialog nodes](#) for a given word or phrase.

## 15 March 2018

### Introducing IBM Watson® Assistant

IBM Watson® Conversation has been renamed. It is now called IBM Watson® Assistant. The name change reflects the fact that Watson Assistant is expanding to provide prebuilt content and tools that help you more easily share the virtual assistants you build. Read [this blog post](#) for more details.

### New REST APIs and SDKs are available for Watson Assistant

The new APIs are functionally identical to the existing Conversation APIs, which continue to be supported. For more information about the Watson Assistant APIs, see the [API Reference](#).

### Dialog enhancements

The following features were added to the dialog tool:

- Simple variable name and value fields are now available that you can use to add context variables or update context variable values. You do not need to open the JSON editor unless you want to. See [Defining a context variable](#) for more details.
- Organize your dialog by using folders to group together related dialog nodes. See [Organizing the dialog with folders](#) for more details.
- Support was added for customizing how each dialog node participates in user-initiated digressions away from the designated dialog flow. See [Digressions](#) for more details.

## Search intents and entities

A new search feature has been added that allows you to [search intents](#) for user examples, intent names, or descriptions, or to [search entity](#) values and synonyms.

## Content catalogs

The new [content catalogs](#) contain a single category of prebuilt common intents and entities that you can add to your application. For example, most applications require a general #greeting-type intent that starts a dialog with the user. You can add it from the content catalog rather than building your own.

## Enhanced user metrics

The Improve component has been enhanced with additional user metrics and logging statistics. For example, the Overview page includes several new, detailed graphs that summarize interactions between users and your application, the amount of traffic for a given time period, and the intents and entities that were recognized most often in user conversations.

## 12 March 2018

### New date and time methods

Methods were added that make it easier to perform date calculations from the dialog. See [Date and time calculations](#) for more details.

## 16 February 2018

### Dialog node tracing

When you use the "Try it out" pane to test a dialog, a location icon is displayed next to each response. You can click the icon to highlight the path that your assistant traversed through the dialog tree to arrive at the response. See [Building a dialog](#) for details.

### New API version

The current API version is now `2018-02-16`. This version introduces the following changes:

- A new `include_audit` parameter is now supported on most GET requests. This is an optional boolean parameter that specifies whether the response should include the audit properties (`created` and `updated` timestamps). The default value is `false`. (If you are using an API version earlier than `2018-02-16`, the default value is `true`.) For more information, see the [API Reference](#).
- Responses from API calls using the new version include only properties with non-`null` values.
- The `output.nodes_visited` and `output.nodes_visited_details` properties of message responses now include nodes with the following types, which were previously omitted:
  - Nodes with `type=response_condition`
  - Nodes with `type=event_handler` and `event_name=input`

## 9 February 2018

### Dutch system entities (Beta)

Dutch language support has been enhanced to include the availability of [System entities](#) in beta release. See [Supported languages](#) for details.

## 29 January 2018

- The Watson Assistant REST API now supports new request parameters:
  - Use the `append` parameter when updating a workspace to indicate whether the new workspace data should be added to the existing data, rather than replacing it. For more information, see [Update workspace](#).
  - Use the `nodes_visited_details` parameter when sending a message to indicate whether the response should include additional diagnostic information about the nodes that were visited during processing of the message. For more information, see [Send message](#).

## 23 January 2018

### Unable to retrieve list of workspaces

If you see this or similar error messages when working in the tooling, it might mean that your session has expired. Log out by choosing **Log out** from the **User information** icon, and then log back in.

## 8 December 2017

### Log data access across instances (Premium users only)

If you are a Watson Assistant Premium user, your premium instances can optionally be configured to allow access to log data from workspaces across your different premium instances.

### Copy nodes

You can now duplicate a node to make a copy of it and its children. This feature is helpful if you build a node with useful logic that you want to reuse elsewhere in your dialog. See [Copying a dialog node](#) for more information.

### Capture groups in pattern entities

You can identify groups in the regular expression pattern that you define for an entity. Identifying groups is useful if you want to be able to refer to a subsection of the pattern later. For example, your entity might have a regex pattern that captures US phone numbers. If you identify the area code segment of the number pattern as a group, then you can subsequently refer to that group to access just the area code segment of a phone number. See [Defining entities](#) for more information.

## 6 December 2017

### IBM Cloud® Functions integration (Beta)

Call IBM Cloud® Functions (formerly IBM OpenWhisk) actions directly from a dialog node. This feature enables you to, for example, call an action to retrieve weather information from within a dialog node, and then condition on the returned information in the dialog response. Currently, you can call an action from a Cloud Functions instance that is hosted in the US South region from Watson Assistant instances that are hosted in the US South region. See [Making programmatic calls from a dialog node](#) for more details.

## 5 December 2017

### **Redesigned UI for Intents and Entities**

The Intents and Entities tabs have been redesigned to provide an easier, more efficient workflow when creating and editing entities and intents. See [Defining intents](#) and [Defining entities](#) for information about working with these tabs.

## 30 November 2017

### **Eastern Arabic numeral support**

Eastern Arabic numerals are now supported in Arabic system entities.

## 29 November 2017

### **Improving understanding of user input across workspaces**

You can now improve a workspace with utterances that were sent to other workspaces within your instance. For example, you might have multiple versions of production workspaces and development workspaces; you can use the same utterance data to improve any of these workspaces. See [Improving across workspaces](#).

## 20 November 2017

### **GB18030 compliance**

GB18030 is a Chinese standard that specifies an extended code page for use in the Chinese market. This code page standard is important for the software industry because the China National Information Technology Standardization Technical Committee has mandated that any software application that is released for the Chinese market after September 1, 2001, be enabled for GB18030. The Watson Assistant service supports this encoding, and is certified GB18030-compliant.

## 9 November 2017

### **Intent examples can directly reference entities**

You can now specify an entity reference directly in an intent example. That entity reference, along with all its values or synonyms, is used by the Watson Assistant service classifier for training the intent. For more information, see [Entity as example](#) in the [Intents](#) topic.

Currently, you can only directly reference closed entities that you define. You cannot directly reference [pattern entities](#) or [system entities](#).

## 8 November 2017

## **Watson Assistant connector**

You can use the new Watson Assistant connector tool to connect your workspace to a Slack or Facebook Messenger app that you own, making it available as a chatbot that Slack or Facebook Messenger users can interact with. This tool is available only for the IBM Cloud US South region.

## **3 November 2017**

### **Dialog updates**

The following updates make it easier for you to build a dialog. (See [Building a dialog](#) for details.)

- You can add a condition to a slot to make it required only if certain conditions are met. For example, you can make a slot that asks for the name of a spouse required only if a previous (required) slot that asks for marital status indicates that the user is married.
- You can now choose **Skip user input** as the next step for a node. When you choose this option, after processing the current node, your assistant jumps directly to the first child node of the current node. This option is similar to the existing *Jump to next step* option, except that it allows for more flexibility. You do not need to specify the exact node to jump to. At run time, your assistant always jumps to whichever node is the first child node, even if the child nodes are reordered or new nodes are added after the next step behavior is defined.
- You can add conditional responses for slots. For both Found and Not found responses, you can customize how your assistant responds based on whether certain conditions are met. This feature enables you to check for possible misinterpretations and correct them before saving the value provided by the user in the slot's context variable. For example, if the slot saves the user's age, and uses `@sys-number` in the *Check for* field to capture it, you can add a condition that checks for numbers over 100, and responds with something like, *Please provide a valid age in years*. See [Adding conditions to Found and Not found responses](#) for more details.
- The interface you use to add conditional responses to a node has been redesigned to make it easier to list each condition and its response. To add node-level conditional responses, click **Customize**, and then enable the **Multiple responses** option.

The **Multiple responses** toggle sets the feature on or off for the node-level response only. It does not control the ability to define conditional responses for a slot. The slot multiple response setting is controlled separately.

- To keep the page where you edit a slot simple, you now select menu options to a.) add a condition that must be met for the slot to be processed, and b.) add conditional responses for the Found and Not found conditions for a slot. Unless you choose to add this extra functionality, the slot condition and multiple responses fields are not displayed, which declutters the page and makes it easier to use.

## **25 October 2017**

### **Updates to Simplified Chinese**

Language support has been enhanced for Simplified Chinese. This includes intent classification improvements using character-level word embeddings, and the availability of system entities. Note that the Watson Assistant service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

## Updates to Spanish

Improvements have been made to Spanish intent classification, for very large datasets.

## 11 October 2017

### Updates to Korean

Language support has been enhanced for Korean. Note that the Watson Assistant service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

## 3 October 2017

### Pattern-defined entities (Beta)

You can now define specific patterns for an entity, using regular expressions. This can help you identify entities that follow a defined pattern, for example SKU or part numbers, phone numbers, or email addresses. See [Pattern-defined entities](#) for additional details.

- You can add either synonyms or patterns for a single entity value; you cannot add both.
- For each entity value, there can be a maximum of up to 5 patterns.
- Each pattern (regular expression) is limited to 128 characters.
- Importing or exporting via a CSV file does not currently support patterns.
- The REST API does not support direct access to patterns, but you can retrieve or modify patterns using the `/values` endpoint.

### Fuzzy matching filtered by dictionary (English only)

An improved version of fuzzy matching for entities is now available, for English. This improvement prevents the capturing of some common, valid English words as fuzzy matches for a given entity. For example, fuzzy matching will not match the entity value `like` to `hike` or `bike`, which are valid English words, but will continue to match examples such as `lkie` or `oike`.

## 27 September 2017

### Condition builder updates

The control that is displayed to help you define a condition in a dialog node has been updated. Enhancements include support for listing available context variable names after you enter the `$` to begin adding a context variable.

## 31 August 2017

### Improve section rollback

The median conversation time metric, and corresponding filters, are being temporarily removed from the Overview page of the Improve section. This removal will prevent the calculation of certain metrics from causing the median conversation time metric, and the conversations over time graph, to display inaccurate information. IBM regrets removing functionality from

the tool, but is committed to ensuring that we are communicating accurate information to users.

## Dialog node names

You can now assign any name to a dialog node; it does not need to be unique. And you can subsequently change the node name without impacting how the node is referenced internally. The name you specify is saved as a title attribute of the node in the workspace JSON file and the system uses a unique ID that is stored in the name attribute to reference the node.

## 23 August 2017

### Updates to Korean, Japanese, and Italian

Language support has been enhanced for Korean, Japanese, and Italian. Note that the Watson Assistant service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

## 10 August 2017

### Accent normalization

In a conversational setting, users may or may not use accents while interacting with the Watson Assistant service. As such, an update has been made to the algorithm so that accented and non-accented versions of words are treated the same for intent detection and entity recognition.

However, for some languages like Spanish, some accents can alter the meaning of the entity. Thus, for entity detection, although the original entity may implicitly have an accent, your assistant can also match the non-accented version of the same entity, but with a slightly lower confidence score.

For example, for the word `barrió`, which has an accent and corresponds to the past tense of the verb `barri` (to sweep), your assistant can also match the word `barrio` (neighborhood), but with a slightly lower confidence.

The system will provide the highest confidence scores in entities with exact matches. For example, `barrio` will not be detected if `barrió` is in the training set; and `barrió` will not be detected if `barrio` is in the training set.

You are expected to train the system with the proper characters and accents. For example, if you are expecting `barrió` as a response, then you should put `barrió` into the training set.

Although not an accent mark, the same applies to words using, for example, the Spanish letter `ñ` vs. the letter `n`, such as `uña` vs. `una`. In this case the letter `ñ` is not simply an `n` with an accent; it is a unique, Spanish-specific letter.

You can enable fuzzy matching if you think your customers will not use the appropriate accents, or misspell words (including, for example, putting a `n` instead of a `ñ`), or you can explicitly include them in the training examples.

**Note:** Accent normalization is enabled for Portuguese, Spanish, French, and Czech.

### Workspace opt-out flag

The Watson Assistant REST API now supports an opt-out flag for workspaces. This flag indicates that workspace training data such as intents and entities are not to be used by IBM for general service improvements. For more information, see the [API Reference](#)

## 7 August 2017

### Next and last date interpretation

The Watson Assistant service treats last and next dates as referring to the most immediate last or next day referenced, which may be in either the same or a previous week. See the [system entities](#) topic for additional information.

## 3 August 2017

### Fuzzy matching for additional languages (Beta)

Fuzzy matching for entities is now available for additional languages, as noted in the [Supported languages](#) topic.

### Partial match (Beta - English only)

Fuzzy matching will now automatically suggest substring-based synonyms present in user-defined entities, and assign a lower confidence score as compared to the exact entity match. See [Fuzzy matching](#) for details.

## 28 July 2017

### Updates

This release includes the following updates:

- When you set bidirectional preferences for the tooling, you can now specify the graphical user interface direction.
- The color scheme of the tooling was updated to be consistent with other Watson services and products.

## 19 July 2017

### REST API now supports access to dialog nodes

The Watson Assistant REST API now supports access to dialog nodes. For more information, see the [API Reference](#).

## 14 July 2017

### Slots enhancement

The slots functionality of dialogs was enhanced. For example, `aslot_in_focus` property was added that you can use to define a condition that applies to a single slot only. See [Gathering information with slots](#) for details.

## 12 July 2017

### Support for Czech

Czech language support has been introduced; please see the [Supported languages](#) topic for additional details.

## 11 July 2017

### Test in Slack

You can use the new **Test in Slack** tool to quickly deploy your workspace as a Slack bot user for testing purposes. This tool is available only for the IBM Cloud US South region.

### Updates to Arabic

Arabic language support has been enhanced to include absolute scoring per intent, and the ability to mark intents as irrelevant; please see the [Supported languages](#) topic for additional details. Note that the Watson Assistant service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

## 23 June 2017

### Updates to Korean

Korean language support has been enhanced; please see the [Supported languages](#) topic for additional details. Note that the Watson Assistant service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

## 22 June 2017

### Introducing slots

It is now easier to collect multiple pieces of information from a user in a single node by adding slots. Previously, you had to create several dialog nodes to cover all the possible combinations of ways that users might provide the information. With slots, you can configure a single node that saves any information that the user provides, and prompts for any required details that the user does not. See [Gathering information with slots](#) for more details.

### Simplified dialog tree

The dialog tree has been redesigned to improve its usability. The tree view is more compact so it is easier to see where you are within it. And the links between nodes are represented in a way that makes it easier to understand the relationships between the nodes.

## 21 June 2017

### Arabic support

Language support for Arabic is now generally available. For details, see [Configuring bidirectional languages](#).

### Language updates

The Watson Assistant service algorithms have been updated to improve overall language support. See the [Supported languages](#) topic for details.

## 16 June 2017

### **Recommendations (Beta - Premium users only)**

The Improve panel also includes a **Recommendations** page that recommends ways to improve your system by analyzing the conversations that users have with your chatbot, and taking into account your system's current training data and response certainty.

## **14 June 2017**

### **Fuzzy matching for additional languages (Beta)**

Fuzzy matching for entities is now available for additional languages, as noted in the [Supported languages](#) topic. You can turn on fuzzy matching per entity to improve the ability of your assistant to recognize terms in user input with syntax that is similar to the entity, without requiring an exact match. The feature is able to map user input to the appropriate corresponding entity despite the presence of misspellings or slight syntactical differences. For example, if you define giraffe as a synonym for an animal entity, and the user input contains the terms giraffes or girafe, the fuzzy match is able to map the term to the animal entity correctly. See [Fuzzy matching](#) for details.

## **13 June 2017**

### **User conversations**

The Improve panel now includes a **User conversations** page, which provides a list of user interactions with your chatbot that can be filtered by keyword, intent, entity, or number of days. You can open individual conversations to correct intents, or to add entity values or synonyms.

### **Regex change**

The regular expressions that are supported by SpEL functions like find, matches, extract, replaceFirst, replaceAll and split have changed. A group of regular expression constructs are no longer allowed, including look-ahead, look-behind, possessive repetition and backreference constructs. This change was necessary to avoid a security exposure in the original regular expression library.

## **12 June 2017**

### **Updates**

This release includes the following updates:

- The maximum number of workspaces that you can create with the **Lite** plan (formerly named the Free plan) changed from 3 to 5.
- You can now assign any name to a dialog node; it does not need to be unique. And you can subsequently change the node name without impacting how the node is referenced internally. The name you specify is treated as an alias and the system uses its own internal identifier to reference the node.
- You can no longer change the language of a workspace after you create it by editing the workspace details. If you need to change the language, you can export the workspace as a JSON file, update the language property, and then import the JSON file as a new workspace.

## 6 June 2017

### Learn

A new *Learn about IBM Watson® Assistant* page is available that provides getting started information and links to service documentation and other useful resources. To open the page, click the icon in the page header.

### Bulk export and delete

You can now simultaneously export a number of intents or entities to a CSV file, so you can then import and reuse them for another Watson Assistant application. You can also simultaneously select a number of entities or intents for deletion in bulk.

### Updates to Korean

Korean tokenizers have been updated to address informal language support. IBM continues to work on improvements to entity recognition and classification.

### Emoji support

Emojis added to intent examples, or as entity values, will now be correctly classified/extracted.

Only emojis that are included in your training data will be correctly and consistently identified; emoji support may not correctly classify similar emojis with different color tones or other variations.

### Entity stemming (Beta - English only)

The fuzzy matching beta feature recognizes entities and matches them based on the stem form of the entity value. For example, this feature correctly recognizes `bananas` as being similar to `banana`, and `run` being similar to `running` as they share a common stem form. For more information, see [Fuzzy matching](#).

### Workspace import progress

When you import a workspace from a JSON file, a tile for the workspace is displayed immediately, in which information about the progress of the import is displayed.

### Reduced training time

Multiple models are now trained in parallel, which noticeably reduces the training time for large workspaces.

## 26 May 2017

### New API version

The current API version is now `2017-05-26`. This version introduces the following changes:

- The schema of `ErrorResponse` objects has changed. This change affects all endpoints and methods. For more information, see the [API Reference](#).
- The internal schema used to represent dialog nodes in exported workspace JSON has changed. If you use the `2017-05-26` API to import a workspace that was exported using an earlier version, some dialog nodes might not import correctly. For best results, always import a workspace using the same version that was used to export it.

## 25 May 2017

### Manage context variables

You can now manage context variables in the "Try it out" pane. Click the **Manage context** link to open a new pane where you can set and check the values of context variables as you test the dialog. See [Testing your dialog](#) for more information.

## 16 May 2017

### Updates

This release includes the following updates:

- A **Car Dashboard** sample workspace is now available when you open the tool. To use the sample as a starting point for your own workspace, edit the workspace. If you want to use it for multiple workspaces, then duplicate it instead. The sample workspace does not count toward your subscription workspace total unless you use it.
- It is now easier to navigate the tool. The navigation menu options are available from the side of the main page instead of the top. In the page header, breadcrumb links display that show you where you are. You can now switch between service instances from the Workspaces page. To get there quickly, click **Back to workspaces** from the navigation menu. If you have multiple service instances, the name of the current instance is displayed. You can click the **Change** link beside it to choose another instance.
- When you create a dialog, two nodes are now added to it for you: 1) a **Welcome** node at the start the dialog tree that contains the greeting to display to the user and 2) an **Anything else** node at the end of the tree that catches any user inquiries that are not recognized by other nodes in the dialog and responds to them. See [Creating a dialog](#) for more details.
- When you are testing a dialog in the "Try it out" pane, you can now find and resubmit a recent test utterance by pressing the Up key to cycle through your previous inputs.
- Experimental Korean language support for 5 system entities (@sys-date, @sys-time, @sys-currency, @sys-number, @sys-percentage) is now available. There are known issues for some of the numeric entities, and limited support for informal language input.
- An Overview page is available from the Improve tab. The page provides a summary of interactions with your bot. You can view the amount of traffic for a given time period, as well as the intents and entities that were recognized most often in user conversations. For additional information, see [Using the Overview page](#).

## 27 April 2017

### System entities

The following system entities are now available as beta features in English only:

- sys-location: Recognizes references to locations, such as towns, cities, and countries, in user utterances.
- sys-person: Recognizes references to people's names, first and last, in user utterances.

For more information, see the [System entities reference](#).

### Fuzzy matching for entities

Fuzzy matching for entities is a beta feature that is now available in English. You can turn on fuzzy matching per entity to improve the ability of your assistant to recognize terms in user input with syntax that is similar to the entity, without requiring an exact match. The feature is able to map user input to the appropriate corresponding entity despite the presence of misspellings or slight syntactical differences. For examples, if you define **giraffe** as a synonym for an animal entity, and the user input contains the terms *giraffes* or *girafe*, the fuzzy match is able to map the term to the animal entity correctly. See [Defining entities](#) and search for [Fuzzy Matching](#) for details.

## 18 April 2017

### Updates

This release includes the following updates:

- The Watson Assistant REST API now supports access to the following resources:
  - entities
  - entity values
  - entity value synonyms
  - logs

For more information, see the [API Reference](#).

- The behavior of the `/messages POST` method has changed the handling of entities and intents specified as part of the message input:
  - If you specify intents on input, your assistant uses the intents you specify, but uses natural language processing to detect entities in the user input.
  - If you specify entities on input, your assistant uses the entities you specify, but uses natural language processing to detect intents in the user input.

The behavior has not changed for messages that specify both intents and entities, or for messages that specify neither.

- The option to mark user input as irrelevant is now available for all supported languages. This is a beta feature.
- A new Credentials tab provides a single place where you can find all of the information you need for connecting your application to a workspace (such as the Watson Assistant credentials and workspace ID), as well as other deployment options. To access the Credentials tab for your workspace, click the icon and select **Credentials**.

## 9 March 2017

### REST API updates

The Watson Assistant REST API now supports access to the following resources:

- workspaces
- intents
- examples
- counterexamples

For more information, see the [API Reference](#).

## 7 March 2017

### Intent name restrictions

The use of `.` or `..` as an intent name causes problems and is no longer supported. You cannot rename or delete an intent with this name; to change the name, export your intents to a file, rename the intent in the file, and import the updated file into your workspace. Paying customers can contact support for a database change.

## 1 March 2017

### System entities are now enabled in German

System entities are now enabled in German.

## 22 February 2017

### Messages are now limited to 2,048 characters

Messages are now limited to 2,048 characters.

## 3 February 2017

### Updates

This release includes the following updates:

- We changed how intents are scored and added the ability to mark input as irrelevant to your application. For details, see [Defining intents](#) and search for `Mark as irrelevant`.
- This release introduced a major change to the workspace. To benefit from the changes, you must manually upgrade your workspace.
- The processing of **Jump to** actions changed to prevent loops that can occur under certain conditions. Previously, if you jumped to the condition of a node and neither that node nor any of its peer nodes had a condition that was evaluated as true, the system would jump to the root-level node and look for a node whose condition matched the input. In some situations this processing created a loop, which prevented the dialog from progressing.

Under the new process, if neither the target node nor its peers is evaluated as true, the dialog turn is ended. To reimplement the old model, add a final peer node with a condition of `true`. In the response, use a **Jump to** action that targets the condition of the first node at the root level of your dialog tree.

## 11 January 2017

## Customize node titles

In this release, you can customize node titles in dialog.

## 22 December 2016

### Node title section

In this release, dialog nodes display a new section for `node title`. The ability to customize the `node title` is not available. When collapsed, the `node title` displays the `node condition` of the dialog node. If there is not a `node condition`, "Untitled Node" is displayed as the title.

## 19 December 2016

### Dialog editor UI changes

Several changes make the dialog editor easier and more intuitive to use:

- A larger editing view makes it easier to view all the details of a node as you work on it.
- A node can contain multiple responses, each triggered by a separate condition. For more information see [Multiple responses](#).

## 5 December 2016

### Updates

This release includes the following updates:

- New languages are supported, all in Experimental mode: German, Traditional Chinese, Simplified Chinese, and Dutch.
- Two new system entities are available: `@sys-date` and `@sys-time`. For details, see [System entities](#).

## 21 October 2016

### Updates

This release includes the following updates:

- The Watson Assistant service now provides system entities, which are common entities that can be used across any use case. For details, see [Defining entities](#) and search for `Enabling system entities`.
- You can now view a history of conversations with users on the Improve page. You can use this to understand your bot's behavior. For details, see [Improving your skill](#).
- You can now import entities from a comma-separated value (CSV) file, which helps with when you have a large number of entities. For details, see [Defining entities](#) and search for `Importing entities`.

## 20 September 2016

### New version 2016-09-20

To take advantage of the changes in a new version, change the value of the `version` parameter to the new date. If you're not ready to update to this version, don't change your version date.

- **version 2016-09-20:** `dialog_stack` changed from an array of strings to an array of JSON objects.

## 29 August 2016

### Updates

This release includes the following updates:

- You can move dialog nodes from one branch to another, as siblings or peers. For details, see [Moving a dialog node](#).
- You can expand the JSON editor window.
- You can view chat logs of your bot's conversations to help you understand its behavior. You can filter by intents, entities, date, and time. For details, see [Improving your skill](#).

## 11 July 2016

### General Availability

This General Availability release enables you to work with entities and dialogs to create a fully functioning bot.

## 18 May 2016

### Experimental release

This Experimental release of the Watson Assistant introduces the user interface and enables you to work with workspaces, intents, and examples.

## Web chat release notes

---

Find out what's new in the web chat integration.

The web chat change log lists changes ordered by version number. For more information about the web chat, see [Integrating the web chat with your website](#).

For information about new features and improvements to the core Watson Assistant product, see [Release notes](#).

### Controlling the web chat version

If you want to evaluate changes that are introduced in a web chat release before you apply them to your deployment, you can

set a version of your web chat. For more information, see [Controlling the web chat version](#).

### 7.1.1

Release date: 13 February 2023

- **New journey events:** The new `tour:start`, `tour:end`, and `tour:step` events provide details about the user's progress through a journey (also known as a *tour*). These events can be used to navigate to a specific page when the user starts a journey or reaches a certain step, or to show a survey after a journey ends.
- **New journey instance methods:** The new `tour` object supports instance methods that provide better control over journeys. You can use these methods to start or end a journey, or to automatically navigate through a journey in response to user actions.
- **Added journey strings to the language pack:** New strings for journeys have been added to the language pack. You can modify the strings in the language pack by using the `updateLanguagePack()` instance method. For more information about language packs, see [Languages](#).

For more information about the journeys beta feature, see [Guiding customers with journeys](#).

### 7.1.0

Release date: 17 January 2023

- **Updated Zendesk agent app:** The agent app for Zendesk has been updated for compatibility with Zendesk workspaces.
- In the service desk starter kits, the instance of the web chat integration has been added to the `serviceDeskFactory` [parameters](#) to make it accessible to custom service desk implementations.
- **New instance methods:** The new `elements.getMessageInput()` and `elements.getHomeScreenInput()` instance methods enable access to the input fields used by the customer to send messages. You can use these methods to change the input or to take action as the user is typing (for example, to implement for a type-ahead feature).
- **New event:** The new `agent:pre:sessionHistory` event enables you to filter potential PII from messages sent from a customer or service desk agent before the messages are sent to the assistant for storage in the session history.
- **New property in web chat state object:** In the object returned from the `getState()` instance method, the new `isDebugEnabled` property indicates whether the web chat debug flag is set to `true`.

### 7.0.0

Release date: 5 December 2022

- **Streamlined live agent handoff:** The live agent handoff experience has been streamlined and simplified. Instead of opening the live agent chat in a separate window, the web chat now shows the live agent entering the conversation in the same window.

Because of this change, the `updateCustomMenuOptions` instance method has changed to reflect the fact that there is now only a single view, with a single list of custom menu options. If you want to customize menu options only during a live agent chat, you can subscribe to the `agent:pre:startChat` and `agent:endChat` events to trigger your customizations.

- **agent:endChat changes:** The `agent:endChat` event now also fires if the customer cancels a live agent request before the agent has joined. If you want to show a post-chat form only after a live agent chat, you can use the new `requestCancelled` flag on the event to determine whether the request was canceled.

- **New configuration options:** The following new options are available in the configuration object:
  - `serviceDesk.availabilityTimeoutSeconds`: Specifies how long the web chat waits for an available agent before automatically canceling the live agent request.
  - `serviceDesk.disableAgentSessionHistory`: Disables storage of live agent chats in the session history. If this option is set to `true`, live agent chat history is not stored; this means that if the web chat is reloaded, the live agent chat history is lost.
- For more information, see [Service desk options](#).
- **elements instance property:** A new `elements` instance property provides methods you can use to apply CSS styles to individual elements used by the web chat. (Currently, only the main window is supported.)
- **skip\_card option for journeys:** The journeys beta feature has been updated to support a new `skip_card` property. You can use this property to start a journey immediately without waiting for the customer to click the introductory card, or even to start a journey from your website without opening the web chat at all. For more information, see [Guiding customers with journeys](#).
- **Path changes:** Some internal paths used for communication with the assistant have changed. If you have firewall or proxy rules that are configured to allow specific paths, you might need to update your configuration to allow the following paths:
  - `/<SUBSCRIPTION_ID>/chat/<INTEGRATION_ID>/config`
  - `/<SUBSCRIPTION_ID>/chat/<INTEGRATION_ID>/message`

## 6.9.0

Release date: 14 November 2022

- You can now create *journeys* to guide your customers through tasks they can already complete on your website. A journey is an interactive, multipart response that can combine text, video, and images, presented in sequence in a small window superimposed over your website.

Journeys are available as a beta feature. For more information, see [Guiding customers with journeys](#).

## 6.8.1

Release date: 7 November 2022

- Bug fixes.

## 6.8.0

Release date: 31 October 2022

- Added support for sending pre-chat information to the Salesforce integration.

## 6.7.0

Release date: 10 October 2022

- **New `updateIsTypingCounter()` method:** The new `updateIsTypingCounter()` instance method updates the counter that determines whether the typing indicator is displayed. For more information, see [instance.updateIsTypingCounter\(\)](#).
- **New `updateBotUnreadIndicatorVisibility()` method:** The new `updateBotUnreadIndicatorVisibility()` instance method specifies whether the unread indicator on the launcher icon is shown or hidden. For more information, see [instance.updateBotUnreadIndicatorVisibility\(\)](#).

- Connect to Agent and custom cards now have rounded corners.
- Bug fixes.

## 6.6.2

*Release date: 15 August 2022*

- Bug fixes

## 6.6.1

*Release date: 8 August 2022*

- The `servers` property now supports a new `webChatScriptPrefix` option. Use this property to configure a proxy between your users' browsers and the IBM Cloud servers that host the web chat JavaScript code. For more information, see [Setting up a proxy](#).

## 6.6.0

*Release date: 25 July 2022*

- A new `servers` property is now available in the web chat configuration options. You can use this property to set up a proxy between your users' browsers and Watson Assistant. For more information, see [Setting up a proxy](#).

## 6.5.2

*Release date: 11 July 2022*

- Bug fix for `date` response type.

## 6.5.1

*Release date: 15 June 2022*

- Bug fix for the Zendesk integration.

## 6.5.0

*Release date: 6 June 2022*

- **New agent events:** New events are now fired by the web chat when messages are sent or received during a conversation with a human agent using a service desk integration. For more information, see [Agent events summary](#).
- Bug fixes.

## 6.4.1

*Release date: 16 May 2022*

- **Minimum size:** The minimum allowed size of the rendered web chat window has been reduced to satisfy the accessibility requirements defined by the [Web Content Accessibility Guidelines \(WCAG\) 2.1](#) standard.
- **Pop-up windows and tabs from iframes:** The web chat now allows pop-up windows and new tabs to be opened from content rendered inside `iframe` responses.
- **Faster responses:** Responses received from the assistant are now displayed more quickly and without a... typing

indicator.

## 6.4.0

Release date: 18 April 2022

- **Date picker:** If you configure a step to collect a **Date** customer response, the step now uses the new `date` response type to request that the web chat display a graphical date picker the customer can use to select a date, as an alternative to typing the date in the input field. Existing steps do not automatically inherit this behavior; if you want to use the date picker, you must delete the existing Date response and then re-add it.
- **Skip "connect to agent" card:** A new `serviceDesk.skipConnectAgentCard` configuration option is available. If this option is enabled, the web chat immediately connects to an agent when it receives a *Connect to agent* response, without first displaying a card and waiting for the user to click.
- **Close button:** A new `showCloseAndRestartButton` configuration option specifies whether the web chat interface shows an X (Close) button in addition to the existing - (Minimize) button. A customer can click this button to close the web chat, end the conversation, and end any conversation with a human agent. The chat transcript is also cleared, but any transcript of a conversation with a human agent is preserved.

## 6.3.0

Release date: 24 March 2022

- **Search cards:** Search cards have a new design.
- **New `restartConversation()` method:** The new `restartConversation()` instance method restarts the conversation with the assistant by clearing the web chat transcript and starting a new session. It also fires two new events (`pre:restartConversation` and `restartConversation`).

For more information, see [instance.restartConversation\(\)](#), [pre:restartConversation](#), and [restartConversation](#).

- **New `agentEndConversation()` method:** The new `agentEndConversation()` instance method immediately ends the conversation with a human agent without requesting confirmation from the user. For more information, see [instance.agentEndConversation\(\)](#).
- Bug fixes.

## 6.2.0

Release date: 7 March 2022

- **Navigation:** The web chat has been updated with new navigation features. For example, new “Back” and “Minimize” buttons make it easier to navigate between the home screen, the chat view, and panels. A new customizable drop-down menu appears near the avatar in both the assistant and agent chat views. This new functionality allows you to easily add new options to the menu of the web chat (for more information, see [updateCustomMenuOptions](#)).

The experience of connecting to a human agent has also been improved to make it clearer how a user requests an agent, returns to chatting with the assistant, and ends the conversation.

The animations for the web chat panels have also been improved to make the whole experience more seamless and cohesive.

- **Launcher:** The new web chat launcher bounces on two different occasions to attract attention and encourage customer

engagement. For more information, see [Launcher appearance and behavior](#).

- **Launcher:** Support has been added to control what text the launcher greets the user with, and when the greeting message is shown. For more information, see [Launcher](#).
- **Locale:** The web chat no longer sets the locale in the system context to `en-us` when no locale is configured. The locale is set in the system context only if it is configured for web chat.
- Bug fixes.

## 6.1.0

*Release date: 7 February 2022*

- Updated to support internal changes to the preview link feature.

## 6.0.1

*Release date: 24 January 2022*

- Bug fix for the disclaimer. For more information about the disclaimer, see [Configuration options object](#).

## 6.0.0

*Release date: 19 January 2022*

- **API version:** The web chat now uses the `2021-11-27` version of the Watson Assistant API. Previously it used the `2020-09-24` API version. For information about API changes that have been introduced since the `2020-09-24` version, see the release notes for [27 November 2021](#) and [16 July 2021](#).
- **Launcher:** The new web chat launcher welcomes and engages customers so they know where to find help if they need it. For more information, see [Launcher appearance and behavior](#).
- **Home screen:** The web chat home screen has been updated to have a more modern look. For more information about the home screen, see [Configuring the home screen](#).
- **Agent events:** New events are now fired by the web chat when interacting with a human agent using a service desk integration. If you are using a custom service desk integration based on the [starter kit](#), you can use these events to create a pre-chat form before the agent escalation occurs, to create a post-chat form after the agent conversation ends, or to specify what happens if an agent isn't available (like create a ticket submission form). For more information, see [Agent events summary](#).
- **Markdown support:** The web chat now fully supports common Markdown formatting in messages received from an assistant. You might need to review existing assistant output that contains strings that might be recognized as Markdown. (For example, a line of text that begins with a greater-than (`>`) character is interpreted as a block quote.)
- **Time zone:** The time zone set in the context by the web chat no longer overrides any time zone set by the assistant.
- **Locale:** Any locale configured for the web chat is now sent to the assistant as part of the context.
- **Window open events:** The `window:pre:open` and `window:open` events now fire any time the chat window is opened, regardless of the reason. In previous releases, these events only fired if the window was opened by the customer clicking on the built-in launcher. Other methods of opening the chat window, such as session history or custom launchers, did not fire these events.

The event data passed to the listener has a new `reason` property that indicates the reason the window was opened. If you want to preserve the previous behavior, you can modify your handler to check this property:

```
$ instance.on({ type: "window:open", handler: event => {
  if (event.data.reason === 'default_launcher') {
    // Previous code.
  }
}});
```

For more information, see [Window open reasons](#).

- **hideCloseButton property renamed:** The `hideCloseButton` property for custom panels has been renamed `hideBackButton`. The behavior of the property has not changed. For more information, see [customPanel.open\(\)](#).

## 5.1.2

*Release date: 11 December 2021*

- Bug fix for Salesforce integration.

## 5.1.1

*Release date: 5 November 2021*

- **"User is typing" support:** The web chat now supports displaying the "user is typing" message for service desks. This feature is supported for the Salesforce and Zendesk integrations, as well as any [starter kit](#) integration that implements it.
- Bug fixes.

## 5.1.0

*Release date: 28 October 2021*

- **Custom Panels:** The web chat now supports customizable panels you can use to display any custom HTML content (for example, a feedback form or a multistep process). Your code can use instance methods to dynamically populate a custom panel, as well as open and close it. For more information, see [Custom Panels](#).

## 5.0.2

*Release date: 4 October 2021*

- A [new tutorial](#) is now available that shows how to use Carbon components to customize user-defined responses and writeable elements.
- Bug fixes.

## 5.0.1

*Release date: 20 September 2021*

- Bug fixes.

## 5.0.0

*Release date: 16 September 2021*

- **New response types:** The web chat now supports the new `video`, `audio`, and `iframe` response types. For more information about these response types, see [Rich responses](#).
- **Link to start web chat:** You can now create a set of HTML links that go directly to your web chat and start conversations on

specific topics. For example, you might want to send an email inviting customers to update their account information; you can include a link that opens the web chat on your site and sends the initial message `I want to update my account`. For more information, see [Creating links to web chat](#).

- **CSS improvements:** Support for CSS styles has been improved to change the way the web chat resets styles in areas where you can include your own custom content, such as user-defined responses and writeable elements. The new approach better protects custom content from accidental style overrides. For more information about custom content and CSS classes, see [Theming & custom content](#).

 **Note:** If you have any custom content (such as user-defined responses or writeable elements), verify that any styling is still rendering as you expect. Consider using the new `ibm-web-chat--default-styles` class to maintain consistency with the web chat default styles.

- **Support for Carbon components:** As part of the new styling support, you can now use [Carbon components](#) in user-defined responses and web chat writeable elements. These components will inherit any theming customizations you have made to the web chat.
- **New embedded script:** The embedded script you use to add the web chat to your website has been updated to avoid unexpected code changes when you lock on to a web chat version. (For more information about web chat versioning, see [Versioning](#).) The previous version of the script will continue to work but is now deprecated. If you want to upgrade your existing web chat deployments to use the new script, copy the updated code snippet from the **Embed** tab of the web chat integration settings. (Remember to reapply any customizations you have made.)
- **Removal of deprecated methods and events:**
  - The `error` event has been replaced by the `onError` method in the [configuration object](#).
  - The `getID` method has been removed.
- Microsoft Internet Explorer 11 is no longer a supported browser.

## 4.5.1

*Release date: 30 August 2021*

- Bug fixes for the interactive launcher beta feature. (For more information about this feature, see the `launcherBeta` configuration option at [Configuration options object](#).)

## 4.5.0

*Release date: 29 July 2021*

- A new `scrollToMessage` method is available for scrolling the web chat view to a specified message in the chat history. For more information, see [instance.scrollToMessage](#).
- A new `pre:open` event is available. This event is fired when the web chat window is opened, but before the welcome message or chat history are loaded. For more information, see [window:pre:open](#).
- A new chat history widget is available for embedding in service desk agent UIs. This new widget is based on a read-only view of the standard web chat widget. For information about using the new chat history widget in integrations built using the starter kit, see [Embedded agent application](#).

## 4.4.1

*Release date: 6 July 2021*

- Bug fixes.

## 4.4.0

Release date: 25 June 2021

- Bug fixes.

## 4.3.0

Release date: 7 June 2021

- **Search suggestions:** If a search skill is configured for your assistant, the suggestions include a new **View related content** section. This section contains search results that are relevant to the user input.
- **Focus trap:** A new `enableFocusTrap` option enables maintaining focus inside the web chat widget while it is open. For more information, see [Configuration options object](#).

## 4.2.1

Release date: 6 May 2021

- **Service URLs updated:** The URLs used by the web chat to communicate with the Assistant service have been updated to remove the dependency on the deprecated `watsonplatform.net` domain. This change applies retroactively to version 3.3.0 and all subsequent web chat releases. Make sure the system that hosts the web chat widget has access to the new URL; for more information, see [Deploy your assistant in production](#).

## 4.2.0

Release date: 27 April 2021

- **Conversation starters in suggestions:** The conversation starters you configure for the home screen are now shown as suggestions. If suggestions are enabled, the conversation starters appear in a new section titled **People are also interested in**. This provides an easy way for customers to change the subject or start the conversation over. For more information about the suggestions feature, see [Showing more suggestions](#). For more information about the home screen, see [Configuring the home screen](#).
- **onError callback:** The new `onError` callback option in the web chat configuration enables you to specify a callback function that is called if errors occur in the web chat. This makes it possible for you to handle any errors or outages that occur with the web chat. For more information, see [Listening for errors](#).
- **Session ID available in widget state:** The state information returned by the `getState()` instance method now includes the session ID for the current conversation. For more information, see [instance.getState\(\)](#).
- **IBM watermark:** The web chat can now display a **Built with IBM Watson** watermark to users. This watermark is always enabled for any new web chat integrations on Lite plans. For more information, see [Create a web chat instance to add to your website](#).
- **Fixes to rendering of list items:** The rendering of HTML list items in the web chat widget has been updated.

## 4.1.0

Release date: 8 April 2021

- **Home screen now generally available:** Ease your customers into the conversation by adding a home screen to your web chat window. The home screen greets your customers and shows conversation starter messages that customers can click to easily start chatting with the assistant. For more information about the home screen feature, see [Configuring the home screen](#).

- **Home screen enabled by default** The home screen feature is now enabled by default for all new web chat deployments.
- **Home screen context support**: You can now access context variables from the home screen. Note that initial context must be set using a `conversation_start` node. For more information, see [Starting the conversation](#).

## 4.0.0

Release date: 16 March 2021

- **Session history now generally available**: Session history allows your web chats to maintain conversation history and context when users refresh a page or change to a different page on the same website. It is enabled by default. For more information about this feature, see [Session history](#).

Session history persists within only one browser tab, not across multiple tabs. The dialog provides an option for links to open in a new tab or the same tab. See [this example](#) for more information on how to format links to open in the same tab.

Session history saves changes that are made to messages with the `pre:receive event` so that messages still look the same on rerender. This data is only saved for the length of the session. If you prefer to discard the data, set `event.updateHistory = false;` so the message is rerendered without the changes that were made in the `pre:receive` event.

[instance.updateHistoryUserDefined\(\)](#) provides a way to save state for any message response. With the state saved, a response can be rerendered with the same state. This saved state is available in the `history.user_defined` section of the message response on reload. The data is saved during the user session. When the session expires, the data is discarded.

Two new history events, `history:begin` and `history:end` announce the beginning and end of the history of a reloaded session. These events can be used to view the messages that are being reloaded. The `history:begin` event allows you to edit the messages before they are displayed.

See this example for more information on saving the state of `customResponse` types in session history.

- **Channel switching**: You can now create a dialog response type to functionally generate a connect-to-agent response within channels other than web chat. If a user is in a channel such as Slack or Facebook, they can trigger a channel transfer response type. The user receives a link that forwards them to your organization's website where a connection to an agent response can be started within web chat. For more information, see [Adding a Channel transfer response type](#).

## 3.5.0

Release date: 17 February 2021

- **Session history (beta)**: Web chat session history (beta) is now available. This feature makes it possible to maintain conversation history and context when customers refresh the page or navigate to a different page on the same website. For more information about this beta feature, see [Session history \(beta\)](#).

## 3.4.1

Release date: 2 February 2021

- Made an accessibility enhancement to the chat history. Now, you can use keys to navigate the messages by clicking the chat history, pressing Enter, and then using the arrow keys to move from one message to the next.
- The `instance.updateAssistantInputFieldVisibility()` method was added. You can use it to hide or show the text input field. For example, you might use the `pre:receive` event to check whether an options response type is being returned and if so, hide the text field so the user is forced to pick one of the available options only.
- The `instance.getState()` method was added. You can use it to check for specific conditions, such as `isWebChatOpen`,

before you perform an action that might rely on the condition being true.

For more information about the new methods, see [Instance methods](#).

### 3.3.2

*Release date: 17 December 2020*

- Addressed accessibility issues.

### 3.3.1

*Release date: 3 December 2020*

- The translated strings in the [language files](#) were revised and improved.
- An error message is shown now if a Java Web Token (JWT) that is provided with an incoming message is invalid. If the first JWT fails when the web chat opens, an error message is displayed in place of the web chat window that says, There was an error communicating with Watson Assistant. If the initial JWT is valid, but the token for a subsequent message is invalid, a more discreet error message is displayed in response to the insecure input.
- Bug fixes.

### 3.3.0

*Release date: 23 November 2020*

- Added support for passing contextual information to a service desk agent from web chat.
- You can now customize a `user_defined` response type. For more information, see the [Custom response type tutorial](#).
- Bug fixes.

### 3.2.1

*Release date: 2 November 2020*

- **Bug fix:** Fixing a bug that prevented the web chat integration preview from working after security was enabled.

### 3.2.0

*Release date: 26 October 2020*

- **Security improvement:** If you enable security, you no longer need to include the `identityToken` property when the web chat is loaded on a web page. If a token is not initially provided, the existing `identityTokenExpired` event will be fired when the web chat is first opened to obtain one from your handler.
- **Starter kit update:** The starter kit now allow you to customize the timeout that occurs when the web chat integration checks whether any service desk agents are online.

### 3.1.1

*Release date: 22 October 2020*

- **Accessibility improvement:** Changed how the announcement text is generated to prevent announcements from being duplicated. Announcement text is hidden text that is provided for use by screen readers to indicate when dynamic web page changes occur.

### 3.1.0

*Release date: 8 October 2020*

- **Suggestions now allow for trial and error:** If customers select a suggestion and find that the response is not helpful, they can open the suggestions list again and try a different suggestion.

## 3.0.0

Release date: 22 September 2020

- **Choose when a link to support is included in suggestions:** The Suggestions beta feature has moved to its own tab. Now you can enable suggestions even if your web chat is not set up to connect to a service desk solution. That's because now you can control if and when the option to connect to customer support is available from the suggestions list. For more information, see [Showing more suggestions](#).
- **Search result format change:** To support the ability to show more than 3 search results in a response, the search skill response type format changed. If you are using `pre:receive` or `receive` handlers to process search results, you might need to update your code. The `results` property was replaced by the `primary_results` and `additional_results` properties. For more information about the new search skill response type format, see the [API reference](#).
- **Language pack key change:** Due to improvements that were made to allow you to specify separate chat transfer messages for situations where agents are available and unavailable, the `language source file` was updated. The `agent_chatDescription` was renamed to `default_agent_availableMessage` and another key (`default_agent_unavailableMessage`) was added. If you defined a custom string for the `agent_chatDescription` key, you must modify your code to reflect this change. For more information about the new availability messages and how they are used, see [Adding a Connect to human agent response type](#).

## 2.4.0

Release date: 2 September 2020

- **Add a home screen:** Ease your customers into the conversation by adding a home screen to your web chat window. The home screen greets your customers and shows conversation starter messages that customers can click to submit to the assistant. For more information about this beta feature, see [Adding a home screen](#).

## 2.3.0

Release date: 10 August 2020

- **Introducing Suggestions:** Enable this beta feature to show a helper icon in the chat that customers can click to see alternate topics or to connect to an agent. For more information, see [Showing more suggestions](#).
- **Search results are now expandable:** When search results are displayed in the web chat, users can click **Show more** to see more of the search result text.

## 2.2.0

Release date: 29 July 2020

- **Introduced the `instance.writeableElements()` method:** The `instance.writeableElements()` method gives you zones in the web chat user interface where you can embed your own content. For example, you can add content to the end of the header where it is displayed even as the chat content changes. Or you can add custom content that is displayed before the welcome node. For more information, see [Instance methods](#).
- **Gave the `Send` button a new look**

Changed the icon from  to .

- **Securing your web chat is easier.** You no longer need to specify the `acr` claim when you define the JWT. The Authentication Context Class reference is managed by the web chat automatically.
- Improved the quality of non-English translations.
- Made a few minor bug fixes.

## 2.1.2

Release date: 2 July 2020

- **Fixed a loading issue:** Addressed an issue that prevented the web chat from loading properly for some deployments with short JWT expiration claims.

## 2.1.1

Release date: 1 July 2020

- **Service desk agent initials are displayed:** When the web chat transfers a user to a service desk agent, the agent's avatar is displayed in the chat window to identify messages sent from the service desk agent. If the agent does not have an avatar, the first initial of the agent's given name is displayed instead.

## 2.0

Release date: 16 June 2020

- **Versioning was added to web chat** For more information, see [Versioning](#).
- **Added bidirectional support:** You can now use the `direction` parameter to choose whether to show text and elements in the web chat in right-to-left or left-to-right order. For more information, see [Configuration](#).
- **Introduced the `instance.destroySession()` method:** The `instance.destroySession()` method removes all cookie and browser storage that is related to the current web chat session. You can use this method as part of your website logout sequence. For more information, see [Instance methods](#).

## 1.5.3

Release date: 14 April 2020

- **The `Font family` field was removed from the configuration page:** The text that is displayed in the chat window uses the fonts: IBMplexSans, Arial, Helvetica, sans-serif. If you want to use a different set of fonts, you can customize the CSS for your web chat implementation. For more information, see [Theming](#).
- When your implementation does not specify a unique user ID, the web chat adds a first party cookie with a generated anonymous ID to use to identify the unique user. The generated cookie now expires after 45 days. For more information, see [GDPR and cookie policies](#).

## 1.5.2

Release date: 2 April 2020

- **Introduced the `learningOptOut` parameter:** You can add the `learningOptOut` parameter and set it to `true` to add the `X-Watson-Learning-Opt-Out` header to each `/message` request that originates from the Web Chat. For more information about the header, see [Data collection](#). For more information about the parameter, see [Configuration](#).

## 1.4.0

Release date: 20 March 2020

- **Customize the CSS theme:** For more information, see [Theming](#).
- **Shadow DOM is no longer used:** When you use custom response types or HTML in your dialog, you can apply CSS styles that are defined in your web page to the assistant's response. To override any default styling in the web chat, you must specify the `!important` modifier in your CSS. For more information, see [Rendering response types](#).

# Building your assistant

## Administering your instance

---

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

## Building actions from templates

---

When you create actions, you can choose a template that relates to the problem you're trying to solve. Templates help tailor your actions to include items specific to your business need. The examples in each template can also help you to learn how actions work.

Templates are available for different use cases, for example, booking a meeting, creating a support ticket, or making a payment. Templates include all the pieces that make up an action, such as steps, conditions, and different response types to collect customer answers.



**Note:** You can use templates in English-language assistants only.

Some or all of these features are included as examples in action templates:

Feature	Description	More information
---------	-------------	------------------

Step conditions	A step condition is a Boolean test, based on a runtime value. The step is used only if the test evaluates as true.	<a href="#">Adding conditions to a step</a>
Options synonyms	Synonyms are variations of an option value that customers might enter.	<a href="#">Options</a>
Customized response validation	When you edit a step that expects a customer response, you can customize how validation errors are handled.	<a href="#">When your customer gives invalid answers</a>
Action variables	When customers reply to your assistant, they share information about themselves and what they want. Your assistant remembers this information, and other information about a conversation, as variables.	<a href="#">Using variables to manage conversation information</a>
Session variables	A value that is not necessarily tied to a particular action can be stored as a session variable. Session variables are long-term memory: they persist throughout the user's interaction with the assistant, and your assistant can reference them from any action.	<a href="#">Using variables to manage conversation information</a>
Regex response type	A regex response collects a text string that matches a pattern that is expressed as a regular expression. Use this response to capture a value that must conform to a particular pattern or format, such as an email address or telephone number.	<a href="#">Regex</a>
Yes/No response type	A confirmation response presents customers with the choices of either Yes or No as buttons.	<a href="#">Confirmation</a>
Connect to agent	If your assistant is integrated with one of the supported service desk systems, you can build in logic that transfers the conversation to a live agent when necessary.	<a href="#">Connecting to a live agent</a>
Embedded video	Includes a video to display a how-to demonstration, promotional clip, or other video content. In the web chat, a video response renders as an embedded video player.	<a href="#">Adding a Video response</a>
Customer responses referenced in URLs	In your assistant's output, you can reference variables to personalize a URL link, including information specific to the customer such as account number or email address	<a href="#">Adding and referencing variables</a>
Customized agent handoff information	If your assistant is integrated with one of the supported service desk systems, you can build in logic that transfers the conversation to a live agent when necessary. You can customize messages the assistant displays as part of the transfer	<a href="#">Connecting to a live agent</a>

## Features

### Creating actions from templates

To create actions from templates:

1. Open the **Actions** page.
2. If you have no actions, choose **Create a new action**. If you already have some actions, choose **New action**.
3. On **Create an action**, choose **Quick start with templates**.
4. On **Quick start with templates**, click a template to read details about what it does. Or, you can click the checkbox to select the template right away.
5. Review the details of the template, then click **Select this template**.

6. Your list of selected templates includes all the ones that you chose. You can select as many templates as you want. If you change your mind, click the trash can icon to remove a template from the list.
7. When you're ready, click **Add templates**. Most templates add one action. Some starter kit templates add multiple actions, marked with an asterisk, that require setup. For more information, see [Starter kit extension setup](#).
8. In the actions editor, you can use a new action as-is, or modify it to fit your use case.



**Note:** You can create multiple actions from the same template. For example, if you used the **Book a meeting** template to create an action, you can choose that template again. If the first action is still named **Book a meeting**, the new action is added with the name **Book a meeting (1)**.

## Preview actions

Most action templates are complete. (Starter kit templates are marked with an asterisk and require setup.) You can try out an action right away even without making any changes.

To preview the action that you created from a template:

1. In the actions editor, click **Preview**.
2. Try some of the example phrases from **Customer starts with** to see how the assistant responds.

## Starter kit extension setup

Starter kit templates add actions that can use extensions to connect to data and systems outside of Watson Assistant. When you add a starter kit template, the resulting actions and variables are marked with an asterisk. These require setup of an extension. To learn more about extensions in general, see [Build a custom extension](#).

This table lists each starter kit, a link to download an OpenAPI specification file that you need to set up the extension, and a link to setup instructions.

Starter kit	OpenAPI specification file	Setup instructions
Coveo search	<a href="#">coveo.openapi.json</a>	<a href="#">Coveo search setup</a>
Google custom search	<a href="#">google-custom-search-openapi.json</a>	<a href="#">Google custom search setup</a>
HubSpot	<a href="#">hubspot.advanced.openapi.json</a>	<a href="#">HubSpot setup</a>
NeuralSeek	Requires a file specific to your instance of NeuralSeek. Refer to the setup instructions.	<a href="#">NeuralSeek extension setup</a>
ServiceNow	<a href="#">sn.openapi.json</a>	<a href="#">ServiceNow extension setup</a>
Zendesk	<a href="#">zendesk-openapi.json</a>	<a href="#">Zendesk extension setup</a>

### Starter kits

## Coveo search extension setup

To set up the extension for Coveo search, see [Coveo search extension setup](#).

## Google custom search extension setup

To set up the extension for Google custom search, see [Google custom search extension setup](#).

## HubSpot extension setup

To set up the extension for HubSpot:

1. In HubSpot, you need to create a developer account and then create a private app. The private app includes an access token that you need for authentication. For detailed instructions, see [Getting private app access token](#) in the HubSpot Custom Extension starter kit GitHub repository.
2. Your HubSpot account needs custom properties. Follow the instructions in [Adding Custom Properties in HubSpot](#) in the HubSpot Custom Extension starter kit GitHub repository.
3. Download the OpenAPI specification file: [hubspot.advanced.openapi.json](#).
4. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see [Building the custom extension](#).
5. After you build the HubSpot custom search extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your HubSpot private app access token to authenticate. For general instructions on adding any custom extension, see [Adding an extension to your assistant](#).
6. On the **Actions** page, edit each HubSpot action to use the HubSpot extension in the **And then** section of different steps.

This table explains the edit to each action:

Action	Step Operation	Parameters
*Personalized greeting	2 Get Contact by ID	Set contactId to 1. Hi! Please tell me your customer ID ...
*Dispute a charge	4 Create Ticket	<ul style="list-style-type: none"><li>Set properties.subject to *subject</li><li>Set properties.charge_date to 3. What was the date?</li><li>Set properties.charge_name to 1. Sure, we can help you create a ticket</li><li>Set properties.charge_amount to 2. How much was the charge for?</li><li>Set properties.hs_pipeline_stage to *hs_pipeline_stage</li></ul>
*View all HubSpot tickets	1 List Tickets	Add an expression to set limit to 10
*Check ticket status	8 Get Ticket Info	Set ticketId to *ticketId
*Delete a ticket	2 Archive Ticket	Set ticketId to *ticketId

### HubSpot use extension



**Note:** In the **\*Dispute a charge** action, `subject` and `hs_pipeline_stage` exist by default in HubSpot's ticketing schema. The action adds default values in step 4. The `subject` field in the API corresponds to the ticket name of a HubSpot ticket. The `hs_pipeline_stage` field in the API corresponds to the ticket status of a HubSpot ticket. Submit a

value of `1` for the `hs_pipeline_stage` field, which sets the ticket status to `New`.

## NeuralSeek extension setup

To set up the extension for NeuralSeek, see [NeuralSeek extension setup](#).

## ServiceNow extension setup

To set up the extension for ServiceNow:

1. Request a ServiceNow Personal Developer Instance. For detailed instructions, see [Personal Developer Instances](#).
2. Download the OpenAPI specification file: [sn.openapi.json](#).
3. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see [Building the custom extension](#).
4. After you build the ServiceNow extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your ServiceNow admin username and password to authenticate. Replace the default instance server variable with your own. For general instructions on adding any custom extension, see [Adding an extension to your assistant](#).
5. On the **Actions** page, edit the **\*Create an incident report** action to use the extension. In step 5, click **Edit extension**.
6. In the **Extension** field, choose the ServiceNow extension that you built.
7. In the **Operation** field, choose `Create Incident`.
8. In the **Parameters** list, set:
  - o `impact to *urgency`
  - o `urgency to *urgency`
  - o `assignment_group to *assignment_group`
  - o `short_description to 1. Sure, we can help you create an incident report`

## Zendesk extension setup

To set up the extension for Zendesk:

1. In Zendesk, open the **Admin Center**, for example, <https://{{server-domain}}.zendesk.com/admin>.
2. In **Apps and integrations**, click **Zendesk API**.
3. On the **Settings** tab, enable **Password access**.
4. Download the OpenAPI specification file: [zendesk-openapi.json](#).
5. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see [Building the custom extension](#).
6. After you build the Zendesk extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your Zendesk username and password to authenticate. For general instructions on adding any custom extension, see [Adding an extension to your assistant](#).
7. On the **Actions** page, edit each Zendesk action to use the Zendesk extension in the **And then** section of different steps.

This table explains the edit to each action:

Action	Step Operation	Parameters
*Create a Zendesk support ticket	2 Create Ticket	<ul style="list-style-type: none"> <li>Set <code>request.subject</code> to 3. Thanks! Alright, in just a few words...</li> <li>Set <code>request.comment.body</code> to 4. I'm sorry to hear that ...</li> </ul>
*Update existing ticket	3 Update Ticket	Set <code>id</code> to 1. Let's get your ticket updated...
*View all Zendesk tickets	1 List Tickets	None
*View comments on a ticket	3 Get Ticket Comments	Set <code>id</code> to 2. What's the ticket ID you're looking for?

Zendesk use extension

## Starting the conversation

As you start building your assistant, one of the first things you should consider is how it will start each new conversation with a user. This might be as simple as saying hello, or it might involve asking some questions to gather data the assistant needs before it can do anything else. Starting a new conversation is handled by the *Greet customer* action, which is automatically created (with a default greeting) when you create an assistant.

**Note:** The *Greet customer* action is triggered in situations where your assistant initiates the conversation and then waits for input from the user. Depending on how users connect to your assistant, the Greeting action might not be triggered. (For more information, see [When the greeting action is triggered](#).)

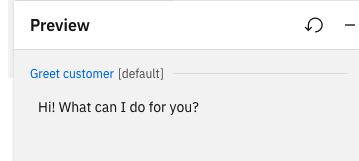
## Customizing the greeting

The *Greet customer* action is automatically provided for any assistant. To customize this action:

1. In the left-hand navigation pane of the actions editor, click **Actions** to expand the list.
2. Click **Set by assistant** to see a list of built-in actions that are automatically provided when you create an assistant.
3. Click *Greet customer* to edit the greeting action.

Notice that for this action, you cannot specify the customer input that starts the action. That's because the greeting action is automatically sent when the assistant starts the conversation, before any user input is received.

4. Under **Conversation steps**, click the first step. In the **Assistant says** field, edit the default text to specify the greeting text you want to use. For example, if you want your assistant to use a more casual tone, you might specify the text `Hi! What can I do for you?`
5. To see your new greeting in action, click **Preview**. In the Preview pane, you should see your customized greeting appear.



## Adding a mandatory welcome flow

Apart from how it is initiated, the *Greet customer* action is just like any other action. If you need to start each conversation with more than just a standard greeting, you can configure your *Greet customer* action with more steps and customer responses, just as you would with any other action. For example, instead of just saying hello, your *Greet customer* action might start by asking for the user's account number. For more information about editing actions, see [Overview: Editing actions](#).

You can also use the *Greet customer* action to initialize variables for use throughout the conversation. For example, you might want to initialize tracking variables at the beginning of the conversation, or store the user's name so you can personalize the assistant's interactions. In general, you can set variables in the *Greet customer* action just as you can with any other action. (For more information about setting variables, see [Managing information during the conversation](#).)

Keep in mind that you should not rely on the *Greet customer* action to initialize required variables unless you are certain it will always be triggered at the beginning of each conversation. For more information, see [When the greeting action is triggered](#).

## When the greeting action is triggered

Depending on how you publish your assistant, the *Greet customer* action might not be triggered. This action is triggered only when the assistant, rather than the user, starts the conversation, as in the following situations:

- Assistant preview
- Phone integration
- Web chat integration with home screen disabled
- A custom client application, depending on design

In this kind of situation, the integration or client application starts the session by sending an empty message to the assistant and waits for the assistant to greet the user (this is when the *Greet customer* action is triggered).

However, there are other situations in which the *Greet customer* action is never triggered:

- Integrations with text messaging channels, such as Slack, Facebook Messenger, or SMS. With these kinds of channels, the user starts the conversation by sending an initial message or request. This triggers the appropriate action for handling the user's request, so the *Greet customer* action is not triggered.
- Web chat integration with home screen enabled. The home screen is an optional feature of the web chat integration. When enabled, the home screen displays a welcome message to the user; because the greeting is defined in the web chat configuration, the *Greet customer* action is not triggered.

## Understanding your users' questions or requests

---

Actions represent the tasks or questions that your assistant can help customers with. Each action has a beginning and an end, making up a conversation between the assistant and a customer. In this topic, learn how to begin an action, where it understands and recognizes a goal based on the words a customer uses to ask a question or make a request.

### Beginning an action

Each assistant can include as many actions as you need to have conversations with your users. You design each individual action to recognize a specific question or request, and when it does, the action starts.

When you create a new action, your first task is to enter one phrase that a customer types or says to start the conversation about a specific topic. This phrase determines the problem your customer has or the question your user asks.

To get going, you only need to enter one phrase, for example: `What are your store hours?`.

After you enter the phrase, it is stored in **Customer starts with**, at the start of the action.

What are your store hours?

Customer starts with:  
What are your store hours?

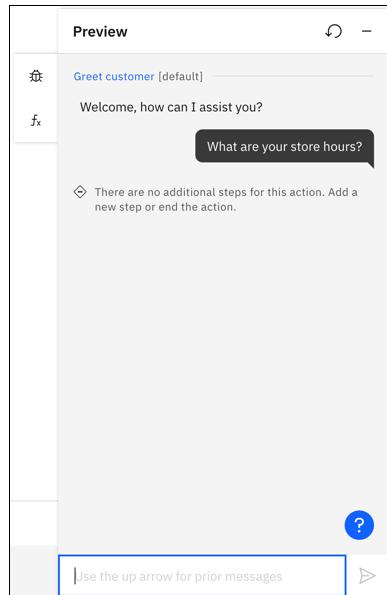
Conversation steps

1 This step has no content  
↓ Continue to next step

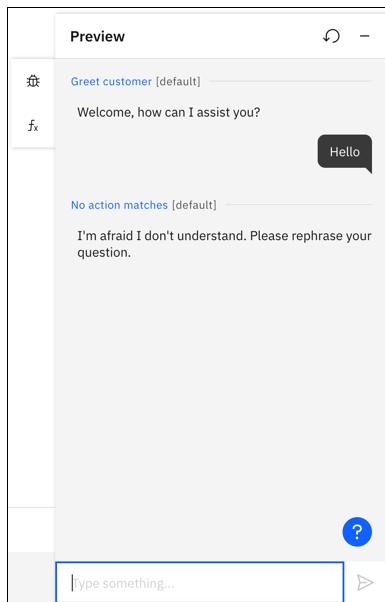
## Testing your phrase

At this point, before even doing anything else with your action, you can already start checking if your assistant recognizes the starting phrase.

1. Click the **Preview** button.
2. Enter your first phrase, for example: `What are your store hours?`.
3. If you see `There are no additional steps for this action`, that means the action recognizes the phrase. (And it's because you haven't added anything else to your action.)



4. If the assistant doesn't understand the phrase, you'll see the built-in action `No action matches`. For more information, see [When the assistant can't understand your customer's request](#).



## Adding more examples

When you're creating a new action, one example phrase is enough to start with. You can build the rest of your action with steps before adding more example phrases. Once doing so, return to **Customer starts with** and add 10 or more variations of the same question or request, using words that your customers commonly use. For example:

- Are you open on the weekend?
- How late are you open today?
- Get store hours
- What time do you open?
- Are you open now?

Each phrase can be up to 1,024 characters in length.

By adding these phrases, your assistant learns when this is the right action for what a customer wants. The additional examples builds the training data that the machine learning engine of Watson Assistant uses to create a natural language processing model. The model is customized to understand your uniquely-defined actions.

## Uploading phrases

If you have a large number of example phrases, you might find it easier to upload them from a comma-separated value (CSV) file than to define them one by one. If you are migrating intent information from the classic Watson Assistant experience to example phrases in the new Watson Assistant experience, see [Migrating intents and entities](#).

1. Collect the phrases into a CSV file. Save the CSV file with UTF-8 encoding and no byte order mark (BOM).

- If you are creating a new CSV file to upload phrases, the format for each line in the file is as follows:

```
$ <phrase>
```

where `<phrase>` is the text of a user example phrase. If you're using a spreadsheet to create a CSV file, put all your phrases into column 1, as shown in the following example:

	A	B
1	Tell me the current weather conditions.	
2	Is it raining?	
3	What's the temperature?	

- If you [downloaded intents from the classic experience](#), the format for each line in the file is as follows:

```
$ <phrase>,<intent>
```

where `<phrase>` is the text of a user example phrase, and `<intent>` is the name of the intent. For example:

```
$ Tell me the current weather conditions.,weather_conditions
Is it raining?,weather_conditions
What's the temperature?,weather_conditions
```

**⚠ Important:** Only one intent can be uploaded per action, so the `<intent>` information listed in the second column of the CSV file must be the same.

2. Go to **Customer starts with** at the start of the action.

3. Click the **Upload** icon .

4. Select a file from your computer.

The file is validated and uploaded, and the system trains itself on the new data.

## Downloading phrases

You can download your example phrases to a CSV file, so you can then upload and reuse them in another Watson Assistant application.

1. Go to **Customer starts with** at the start of the action.

2. Click the **Download** icon .

Your example phrases are downloaded to a CSV file.

## Asking clarifying questions

When your assistant finds that more than one action might fulfill a customer's request, it can automatically ask for clarification. Instead of guessing which action to take, your assistant shows a list of the possible actions to the customer, and asks the customer to pick the right one.



Any **Created by you** action that might match the customer's input can be included in the choices listed by a clarifying question. The **Set by assistant** actions are never included.

In the assistant output, the possible actions are listed by name. The default name for an action is the text of the first example message that you add to it (such as `I want to open an account`), but you can change this name to something more descriptive.

## Customizing clarification

You can control things like the wording your assistant uses to introduce the clarification list.

To customize clarification, complete the following steps:

1. From the **Actions** page, click the **Global settings** icon  in the page header.
2. On the **Ask clarifying question** tab, you can make the following changes:
  - In the **Assistant says** field, edit the text that is displayed before the list of clarification choices. The default text is *Did you mean:*. You can change it to something else, such as *What do you want to do?* or *Pick what to do next*.
  - In the **Label for a fallback choice** field, edit the label that is displayed for the choice that customers can click when none of the other choices are quite right. When a user picks this choice, the *No action matches* system action is taken next.

The label *None of the above* is used if you don't change it.

This fallback choice gives customers a way to get out of the clarification process if it's not helping them. If you don't want to give customers a fallback choice, remove the text from the field.

3. Optionally, review and improve your action names.

If an action name is too long or doesn't reflect the purpose of the action, edit it. Use a name that is concise and represents the overall goal of the action, such as `Open an account` or `Cancel an order`.

## Disabling clarifying questions

You can disable clarifying questions for all actions.

To disable clarification for all actions:

1. From the **Actions** page, click the **Global settings** icon  in the page header.
2. On the **Ask clarifying question** tab, set the switch to **Off**.
3. Click **Save**, and then click **Close**.

## Excluding an action from clarifying questions

You can also prevent a single action from being used in a clarifying question. The effect of this choice depends on the confidence score for the action you exclude.

If the action has the highest confidence score for a customer's question, no clarifying question is asked and the action is triggered.

If the action doesn't have the highest confidence score, the action is excluded from the list of choices in the clarifying question.

For more information about confidence scoring, see [Confidence scoring](#).

To exclude an action from clarification:

1. From the action editor, click the **Action settings** icon 
2. In Action Settings, toggle the **Ask clarifying question** switch to **Off**.

## Coordinating how multiple actions start

As you work on your assistant, it's a good idea to coordinate customer phrase examples across multiple actions. It's important to distinguish how each action is triggered. When a user enters a question or request, the phrase is evaluated across all the **Customer starts with** examples in every action. If two actions have similar phrase examples, then the wrong action might get triggered by your user's question.

## Confidence scoring

Behind the scenes, Watson Assistant determines a confidence score for each phrase. The score is absolute, meaning that a confidence score is assigned based on a predetermined scale, and not relative to other customer phrases. This approach adds flexibility in case multiple questions or requests are detected in a single user input. It also means that the system might not trigger an action at all, if a phrase has a low confidence score. As confidence scores change, your action examples might need restructuring.

To learn more about review and testing confidence scores, see [Action confidence score](#) in [Reviewing and debugging your work](#).

## Adding assistant responses

---

Once an action is activated, the body of the action is composed of multiple steps that make up the conversation between your assistant and your users. One part of each step is what the assistant says to the customer when the step is processed.

To create your assistant's response in a step, you use the **Assistant says** section. This represents the text or speech the assistant delivers to a user at a particular step. Depending on the step, you can add a complete answer to a user's question or ask a follow-up question.

You can enter a simple text response just by entering the text that you want your assistant to display to the user. You can also add formatting and web content, and you can reference user information using *variables*.

## Formatting responses

Use the text editor tools to apply font styling, such as bold or italic, to the text or to add links.

Behind the scenes, font styling and link syntax are stored in Markdown format. If you are using the web chat integration, HTML and Markdown tagging are supported (for more information, see [Markdown formatting](#)).

HTML tags (except for links) are automatically removed from text responses that are sent to the Facebook, WhatsApp, and Slack integrations, because those channels do not support HTML formatting. HTML tags are still handled appropriately in channels that support them (such as the web chat) and stored in the session history.



**Note:** If you're using a custom client application that does not support Markdown, don't apply text styling to your text responses.

## Adding and referencing variables

During the conversation, your assistant stores information as *variables*. Variables are containers for data values that become available at run time; the value of a variable can change over time. Variables include *action variables*, which persist only during a particular action, and *session variables*, which are available to any action. For more information about variables, see [Managing](#)

[information during the conversation.](#)

In your assistant's output, you can reference variables in order to personalize the conversation or include information that is available at run time. For more information about referencing variables in what your assistant says, see [Using variables to customize the conversation](#).

## Testing responses

To see if the assistant responses are formatted correctly, you can use **Preview**.

1. Click the **Preview** button.
2. To start the action, enter your first phrase, for example: `What are your store hours?`.
3. When the assistant responds, check that the message displays as you intended with formatting, use of variables, and so on.

## Tips for adding responses

- Keep answers short and useful.
- Reflect the user's intent in the response. Doing so assures users that the bot is understanding them, or if it is not, gives users a chance to correct a misunderstanding immediately.
- Only include links to external sites in responses if the answer depends on data that changes frequently.
- Word your responses carefully. You can change how someone reacts to your system based simply on how you phrase a response. Changing one line of text can prevent you from having to write multiple lines of code to implement a complex programmatic solution.

## Adding variations

If your users return to your assistant frequently, they might be bored to see the same greetings and responses every time. You can add *response variations* so that your assistant can respond to the same request in different ways.

You can choose to rotate through the response variations sequentially or in random order. By default, responses are rotated sequentially, as if they were chosen from an ordered list.

To add response variations:

1. In **Assistant says**, click the **Add response variations** icon .
2. For **Response variation type**, choose whether to rotate through the response variations sequentially or in random order.  
For more information see [Sequential or random](#).

**Response variations**

Your assistant can respond in different ways to the same topic. [Learn more](#)

Response variation type

Sequential  Random

Response 1

B I @ % %

For example: Please select from the following options:

Response 2

B I @ % %

For example: What type of transfer would you like to make?

Delete

[Add Response +](#)

### Response variations

3. Add each variation into its own field. For example:

Response number	Variation
Response 1	How can I help you?
Response 2	What can I do for you today?
Response 3	Tell me what I can help with.
Response 4	Can I help you?

1. When you're finished, click **Apply**. The variations appear as a block inside **Assistant says**. You can click the **Edit** icon to update the variations, or click the **Delete** icon to remove all the variations. Also, you can add multiple sets of response variations to a step.

**Assistant says**

B I @ % % % %

How can I help you?

4 Edit Delete

### Response variations in Assistant says

## Sequential or random

For **Response variation type**, you can choose **Sequential** or **Random**.

**Sequential** returns the first response variation the first time the action is triggered, the second response variation the second time the action is triggered, and so on, in the same order as you entered the variations. This results in responses being returned in the following order when the node is processed:

- First time:

```
$ How can I help you?
```

- Second time:

```
$ What can I do for you today?
```

- Third time:

```
$ Tell me what I can help with.
```

- Fourth time:

```
$ Can I help you?
```

**Random** selects variation the first time the action is triggered, and randomly selects another variation the next time, but without repeating the same variation consecutively. Here's an example of the order that responses might appear:

- First time:

```
$ Tell me what I can help with.
```

- Second time:

```
$ Can I help you?
```

- Third time:

```
$ How can I help you?
```

- Fourth time:

```
$ What can I do for you today?
```

## Media responses

In addition to text responses, you can use other *response types* to send responses that include multimedia or interactive elements.

The action editor supports the following media response types:

- **Image:** Embeds an image into the response. The source image file must be hosted somewhere and have a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **Video:** Embeds a video player into the response. The source video must be hosted somewhere, either as a playable video on a supported video streaming service or as a video file with a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **Audio:** Embeds an audio clip into the response. The source audio file must be hosted somewhere and have a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **iframe:** Embeds content from an external website, such as a form or other interactive component, directly within the chat. The source content must be publicly accessible using HTTP, and must be embeddable as an HTML `iframe` element.

Different channel integrations have different capabilities for displaying media responses. To see which channel integrations support which response types, see [Channel integration support for response types](#).

If you want to define different responses that are customized for different channels, you can do so by editing the response using

the JSON editor. For more information, see [Targeting specific integrations](#).

By editing your responses using the JSON editor, you can also access additional response types for handling channel-specific interactions.

 **Note:** For more information about how to edit responses using the JSON editor, see [Defining responses using the JSON editor](#).

## Adding an *Image* response

Add an *Image* response to display an image to the customer.

The *Image* response type is supported by the following channel integrations:

- Web chat
- SMS
- Slack
- Microsoft Teams
- Facebook
- WhatsApp

To add an *Image* response, complete the following steps:

1. In the **Assistant says** field, click the  **Image** icon.
2. In the **Source URL** field, type the full URL to the hosted image.

The image must be in `.jpg`, `.gif`, or `.png` format. The image file must be stored in a location that is publicly addressable by an `https:` URL (such as `https://www.example.com/assets/common/logo.png`).

To access an image that is stored in IBM Cloud® Object Storage, enable public access to the individual image storage object, and then reference it by specifying the image source with syntax like this: `https://s3.eu.cloud-object-storage.appdomain.cloud/your-bucket-name/image-name.png`.

3. Optionally specify an image title, description, and alt text in the fields provided. In the web chat integration, the title and description are displayed along with the image.

 **Note:** References to variables are not supported. Some integration channels ignore titles or descriptions.

4. Click **Apply**.

## Adding an *Audio* response

Add an *Audio* response to include spoken-word or other audible content. In the web chat, an audio response renders as an embedded audio player. In the phone integration, an audio response plays over the phone.

The *Audio* response type is supported by the following channel integrations:

- Web chat
- Phone
- SMS
- Slack
- Facebook
- WhatsApp

To add an *Audio* response, complete the following steps:

1. In the **Assistant says** field, click the  **Audio** icon.
2. In the **Source URL** field, type the full URL to the hosted audio clip:
  - To link directly to an audio file, specify the URL to a file in any standard format such as MP3 or WAV. In the web chat, the linked audio clip will render as an embedded audio player.
  - To link to an audio clip on a supported audio hosting service, specify the URL to the audio clip. In the web chat, the linked audio clip will render using the embeddable player for the hosting service.

 **Note:** Specify the URL you would use to access the audio file in your browser (for example, <https://soundcloud.com/ibmresearch/fallen-star-amped>). You do not need to convert the URL to an embeddable form; the web chat will do this automatically.

You can embed audio hosted on the following services:

- [SoundCloud](#)
- [Mixcloud](#)

3. Optionally specify a title, description, and alt text in the fields provided. In the web chat integration, the title and description are displayed along with the audio player.

 **Note:** References to variables are not supported. Some integration channels ignore titles or descriptions.

## Adding a *Video* response

Add a *Video* response to display a how-to demonstration, promotional clip, or other video content. In the web chat, a video response renders as an embedded video player.

The *Video* response type is supported by the following channel integrations:

- Web chat
- SMS
- Slack
- Facebook
- WhatsApp

To add a *Video* response, complete the following steps:

1. In the **Assistant says** field, click the  **Video** icon.
2. In the **Source URL** field, type the full URL to the hosted video:
  - To link directly to a video file, specify the URL to a file in any standard format such as MPEG or AVI. In the web chat, the linked video will render as an embedded video player.
  - To link to a video hosted on a supported video hosting service, specify the URL to the video. In the web chat, the linked video will render using the embeddable player for the hosting service.

 **Note:** HLS (.m3u8) and DASH (MPD) streaming videos are not supported.

**Note:** Specify the URL you would use to view the video in your browser (for example, <https://www.youtube.com/watch?v=52bpMKVigGU>). You do not need to convert the URL to an embeddable form; the web chat will do this automatically.

You can embed videos hosted on the following services:

- [YouTube](#)
- [Facebook](#)
- [Vimeo](#)
- [Twitch](#)
- [Streamable](#)
- [Wistia](#)
- [Vidyard](#)

3. Optionally specify a video title, description, and alt text in the fields provided. In the web chat integration, the title and description are displayed along with the video player.

**Note:** References to variables are not supported. Some integration channels ignore titles or descriptions.

4. If you want to scale the video to a specific display size, specify a number in the **Base height** field.

## Adding an *iframe* response

Add an *iframe* response to embed content from another website directly inside the chat window as an HTML `iframe` element. An *iframe* response is useful if you want to enable customers to perform some interaction with an external service without leaving the chat. For example, you might use an *iframe* response to display the following within the web chat:

- An interactive map on [Google Maps](#)
- A survey using [SurveyMonkey](#)
- A form for making reservations through [OpenTable](#)
- A scheduling form using [Calendly](#)

In the web chat, an *iframe* response renders as a preview card that describes the embedded content. Customers can click this card to display the frame and interact with the content.

The *iframe* response type is supported by the following channel integrations:

- Web chat
- Facebook

To add an *iframe* response type, complete the following steps:

1. In the **Assistant says** field, click the  **iframe** icon.
2. Add the full URL to the external content in the **iframe source** field.

The URL must specify content that is embeddable in an HTML `iframe` element. Different sites have different restrictions for embedding content, and different processes for generating embeddable URLs. An embeddable URL is one that can be specified as the value of the `src` attribute of the `iframe` element.

For example, to embed an interactive map using Google Maps, you can use the Google Maps Embed API. (For more information, see [The Maps Embed API overview](#).) Other sites have different processes for creating embeddable content.

For technical details about using `Content-Security-Policy: frame-src` to allow embedding of your website content,

see [CSP: frame-src](#).

3. Optionally add a descriptive title in the **Title** field.

In the web chat, this title will be displayed in the preview card that the customer clicks to render the external content. (If you do not specify a title, the web chat will attempt to retrieve metadata from the specified URL and display the title of the content as specified at the source.)

 **Note:** References to variables are not supported.

## Technical details: <iframe> sandboxing

Content loaded in an iframe by the web chat is *sandboxed*, meaning that it has restricted permissions that reduce security vulnerabilities. The web chat uses the `sandbox` attribute of the `iframe` element to grant only the following permissions:

Permission	Description
<code>allow-downloads</code>	Allows downloading files from the network, if the download is initiated by the user.
<code>allow-forms</code>	Allows submitting forms.
<code>allow-scripts</code>	Allows running scripts, but <i>not</i> opening pop-up windows.
<code>allow-same-origin</code>	Allows the content to access its own data storage (such as cookies), and allows only very limited access to JavaScript APIs.

 **Note:** A script running inside a sandboxed iframe cannot make changes to any content outside the iframe, *if* the outer page and the iframe have different origins. Be careful if you use an `iframe` response to embed content that has the same origin as the the page where your web chat widget is hosted; in this situation the embedded content can defeat the sandboxing and gain access to content outside the frame. For more information about this potential vulnerability, see the `sandbox` attribute [documentation](#).

## Pause response

Use a *Pause* response to have your assistant wait for a specified interval before displaying the next response. This pause might be to allow time for a request to complete, or simply to mimic the appearance of a live agent who might pause between responses. The pause can be of any duration from 1 to 10 seconds.

A *Pause* response is typically used in combination with other responses. By default, a typing indicator animation appears during the pause in order to simulate a live agent.

The *Pause* response type is supported by the following channel integrations:

- Web chat
- Facebook
- WhatsApp

 **Note:** With the phone channel, you can add a pause by including the SSML `break` element in the assistant output. For more information, see the [Text to Speech documentation](#).

To add a *Pause* response:

1. In the **Assistant says** field, click the  **Pause** icon.
2. In the **Duration** field, enter the length of time for the pause to last as a number of seconds.

The duration can't exceed 10 seconds. Customers are typically willing to wait about 8 seconds for someone to enter a response.

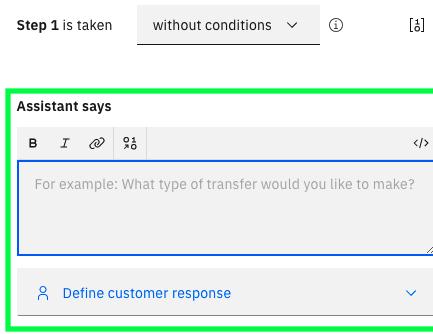
3. The **Typing indicator** is set to **On** by default. You can set this to **Off** if you want.

 **Tip:** Add another response type, such as a text response type, after the pause to clearly denote that the pause is over.

## Collecting information from your customers

Many actions require multiple steps to collect all of the information that is required to complete the customer's request. When a step asks the customer for more information, the *customer response type* defines what type of response is expected.

In the step editor user interface, the middle portion of the step configuration defines the interaction between the assistant and the customer.



Step 1 is taken without conditions ⓘ [x]

Assistant says

B I ⌂ 9:1 ↵

For example: What type of transfer would you like to make?

Define customer response

And then

Continue to next step

Step editor

The **Assistant says** field specifies the output that the assistant sends to the customer. If this output is a question that the user is expected to answer, that answer might be a number, a date, a name, or something else. You use the **Define customer response** field to specify what type of response is expected, based on the kind of information the assistant is asking for and how the customer is expected to specify it.

## Choosing a response type

To choose the customer response type for a step, click **Define customer response** to expand the field. You can then select one of the following response types:

Response type	Description	Example input
<a href="#">Options</a>	A list of predefined choices that customers can select from. At run time, the web chat integration shows an options response as a set of clickable buttons or as a drop-down list, depending on the number of choices.	Small Medium Large

<a href="#">Confirmation</a>	A choice of either Yes or No. At run time, the web chat integration shows the Yes and No options as clickable buttons.	Yes, No
<a href="#">Regex</a>	A text response that matches a specified pattern or format (such as an email address or telephone number).	
<a href="#">Number</a>	A single generic number, specified either as numerals (100) or words (one hundred).	100, one hundred
<a href="#">Date</a>	A single specific date or a range of dates.	31 December 2021, 12/31/2020, tomorrow
<a href="#">Time</a>	A single specific time or a range of time.	5:00 PM, now
<a href="#">Currency</a>	An amount of money, including the unit.	\$25, 500 yen
<a href="#">Percent</a>	A fractional numeric value that is expressed as a percentage.	10%, 50 percent
<a href="#">Free text</a>	Any arbitrary text response.	123 Main Street, John Q. Smith

#### Response types

### **Skipping steps, always asking steps, or never asking steps**

Although a customer response is associated with a particular step, the assistant can recognize the required information at any point during the action. You can set a step to be skipped if its value is already provided in the user's input. If the value is specified after the step, the new value replaces the value that is specified in the step itself.

For example, if the customer's initial input was `I want to withdraw money from my checking account`, a step that asks the user to select a bank account is skipped because the customer already entered that information.

For steps that expect a customer response, you can decide whether to:

- Skip asking if the answer is mentioned in previous messages. This is the default for responses except *Confirmation* and *Free text*.
- Always ask for this information, regardless of previous messages. This is the default for *Confirmation* and *Free text*.
- Never ask. Collect information from previous messages.

### **Always ask**

If your action asks for the same type of data in more than one step, use the **Always ask for this information** setting to prevent the assistant from making incorrect assumptions. For example, you might have an action in which one step asks for a hotel check-in date and another step asks for the check-out date. If you skip asking, the assistant can mistake the check-in date for the check-out date.

To require that a step is always used in the conversation with a customer:

1. In the customer response, click the **Settings** icon to open **Customer response settings**.



2. Choose **Always ask for this information, regardless of previous messages**.

3. Click **Apply**.

## Never ask

There might be some situations where you need a step to never ask a question because you anticipate redundant questions in the conversation.

To set that a step is never asked in the conversation with a customer:

1. In the customer response, click the **Settings** icon to open **Customer response settings**.
2. Choose **Never ask. Collect information from previous messages**.
3. Click **Apply**.

## Example

This example explains when you might set a step to never ask for a response.

You might have an action that responds to requests to file an insurance claim. If you expect customers to typically make a request about a specific type of claim, such as auto, home, or medical, you might not want to ask another question about what type. They might say `I need to file an auto claim` or `I want to make a home claim`.

Although you still need a step that collects the answer about the type of claim, you might not want or need to ask that explicit question, especially if your assistant is used with the phone integration. Instead, you can create a step with the claim options, but set it to never ask.

This table shows how you might set up the steps. The last step is a catch-all in case the customer doesn't mention the claim type initially.

Step	Conditions	Assistant says	Customer response	Customer	And then
				response setting	
1	None	What kind of claim?	Options: Automobile, Homeowner, Medical	Never ask	Continue to the next step
2	Step 1 is Automobile	None	Click here to file an automobile claim	Skip (default)	End the action
3	Step 1 is Homeowner	None	Click here to file a homeowner claim	Skip (default)	End the action
4	Step 1 is Medical	None	Click here to file a medical claim	Skip (default)	End the action
5	Step 1 is not defined (no claim type)	None	Click here to file an insurance claim	Skip (default)	End the action

Example using the never ask response setting

## Customer response types

The configuration information that you must provide varies by response type.

### Options

An *options* response presents customers with a list of choices to select from. How these options are presented depends upon how your customers connect to the assistant. In the web chat integration, the options are shown as clickable buttons (for 4 or fewer options) or as a drop-down list (for more than 4 options).

There are two ways to create the list:

- Enter a list of options and synonyms
- Generate a dynamic list from a variable

### Entering a list of options and synonyms

Enter each choice in the **Option** fields. You can click **Add synonyms** to enter variations of an option value that customers might type. You can enter multiple synonyms in a comma-separated list.

For example, you might define the following options and synonyms:

Option value	Synonyms
Blue	aqua, turquoise, navy
Red	burgundy, crimson, sangria
Green	lime, olive, forest

#### Options example

To select an option, customers can click an option button or list item, such as `Green`. Or they can type `Green` or one of its synonyms, such as `lime`.

Synonyms are useful for a response that might be skipped because they enable the assistant to recognize an option that the customer might have chosen before seeing the list. For example, if the customer's original input was `I want to order a large coffee`, a synonym would enable the assistant to recognize `large` as equivalent to the actual size `Grande`.

**Tip:** You can save this configuration for reuse in other steps. To save a customer response, click the **Save response for reuse** icon. For more information about saved customer responses, see [Saving and reusing customer responses](#).

If you have a long list of options, such as all the states in the United States, you can choose to not show options in a list. This can be useful to prevent a phone integration from reading a long list of options to the customer.

To disable the list:

1. In the options customer response, click the **Settings** icon.
2. Click the **Present options to the customer in a list** toggle to off.

### Dynamic list of options

Within the options customer response, you can use the **dynamic** setting to generate the list when you need to ask questions that are potentially different each time and for each customer. You need to set up a variable as the source of the options. For more

information, see [Dynamic options](#).

## Confirmation

A *confirmation* response presents customers with the choices of either **Yes** or **No** as clickable buttons. Use this response type when the customer's response must be either Yes or No.

The following customer responses are recognized as **Yes**:

- yeah
- yea
- yup
- sure
- positive

The following customer responses are recognized as **No**:

- not
- nope
- nay
- negative

The default setting for *Confirmation* is **Always ask for this information**. For more information, see [Skipping steps, always asking steps, or never asking steps](#).

## Regex

A *regex* response collects a text string that matches a pattern that is expressed as a regular expression. Use this response to capture a value that must conform to a particular pattern or format, such as an email address or telephone number.

You can specify multiple regular expressions for a single response. For example, you might define multiple regex patterns that match part numbers from different vendors that use different formats. Input text for the response is recognized if it matches any of the regex patterns you specify.

To add a regex response:

1. Under **Define customer response** field, click **Regex**.
2. In the **Edit response** window, click in the **Regular expression** field.
3. Select one of the predefined regular expressions, or select **Define custom regular expression** to write your own.

To use a predefined regular expression, select one of the following:

- **Email:** An internet email address (for example, `user@example.com`).
- **Phone number:** A ten-digit US phone number (for example, `800-555-1212` or `(800) 555-1212`).
- **URL:** A correctly formatted URL for an online resource, optionally including the protocol (for example, `example.com` or `https://example.org/index.html`).

 **Tip:** For examples of other common patterns, see [Example regex patterns](#).

To write your own custom regular expression, select **Define custom regular expression** and then type your regex pattern in the **Regular expression** field. For more information on regular expression syntax, see [Syntax](#).

Only English characters can be used in a regular expression. If you need to use other characters in a regular expression, you must represent those characters in Unicode.



**Note:** Watson Assistant uses the Google RE2 regular expression library to match regular expressions at run time.

Regular expression syntax can vary between implementations, so ensure any regex patterns that you write conform to the [RE2 syntax](#).

4. If you want to specify multiple regex patterns for the response, click **Add regular expression** to add another field in which you can select or define another regular expression.

When you specify more than one regular expression, a **Name** field is displayed for each one. Use this field to give each regex pattern a unique name. You can use this name in subsequent step conditions to identify which regex pattern was matched.

Edit response x

Type: Regular expressions [Learn more](#) [Add regular expression +](#)

Name	Regular expression
Email	<code>\b(?:[a-z0-9!#\$%&amp;*+/=?^`{ }~-]+(?:\.[a-z0-9!#\$%&amp;*+/=?^`{ }~-]+)* (?:\.(?:[0-9]{1,3}) [-. ()*](?:[0-9]{1,3})[-. ()*](?:[0-9]{1,4}))?:(?:: *  )</code>
Name	Regular expression
Phone number	<code>\b(?:\+?(?:\d{1,3}))?-.(?:\d{3})\+?(?:\d{3})[-. ]*(?:\d{4})?:(?:: *  )</code>

**Response with multiple regex patterns**

5. Test your regular expression by typing example input in the **Test** field. If any text within your input matches the regex patterns you specified, the matching text is listed in the **Assistant recognizes:** field.

Test ①

My email address is user@example.com, and my phone number is 919-555-6789.]

Assistant recognizes:  
Match 1 [user@example.com](#) Email  
Match 2 [919-555-6789](#) Phone number

**Match shown in regex test**



**Note:** The **Test** feature in the step editor uses a browser-based regex engine to find matches in your test input. At run time, the assistant uses a different regex engine that might have different results, especially with complex patterns. Before deploying your assistant in production, always use the assistant preview to test any step that uses a regex response.



**Tip:** You can save your configured regex response for reuse in other steps. For more information, see [Saving and reusing customer responses](#).

## Example regex patterns

You can use the following regex patterns to recognize some common types of user input.

Description	Patterns
US passport number	<code>/^[A-PR-WY] [1-9] \d\s?\d{4} [1-9]\$/</code>

US bank routing number	\b((0[0-9])\b; (1[0-2])\b; (2[1-9])\b; (3[0-2])\b; (6[1-9])\b; (7[0-2])\b; 80)\b; ([0-9]{7})\b
UPS tracking number	/\b(1Z ?[0-9A-Z]{3} ?[0-9A-Z]{3} ?[0-9A-Z]{2} ?[0-9A-Z]{4} ?\b; [0-9A-Z]{3} ?[0-9A-Z]\b; [\dT]\d\d\d ?\d\d\d\d ?\d\d\d\d)\b/
USPS tracking number	<ul style="list-style-type: none"> <li>• /(\b\d{30}\b)\b; (\b91\d+\b)\b; (\b\d{20}\b) /</li> <li>• /^E\{1\}\d{9}\D{2}\b; ^9\d{15,21}\b/</li> <li>• /^91[0-9]+\$/</li> <li>• /^[A-Za-z]{2}[0-9]+US\$/</li> </ul>
FedEx tracking number	<ul style="list-style-type: none"> <li>• /(\b96\d{20}\b)\b; (\b\d{15}\b)\b; (\b\d{12}\b) /</li> <li>• /\b((98\d\d\d\d?\d\d\d\b; 98\d\d) ?\d\d\d\d ?\d\d\d\d(\ ?\d\d\d\b)?)\b/</li> <li>• /^[0-9]{15}\\$/</li> </ul>

## Example regex patterns

## Number

A *number* response collects a single numeric value.

The customer can specify the number value in either numerals (100) or words (one hundred). Negative and decimal values are recognized.

Date

A **date** response collects a specific calendar date or a range of dates. The assistant can recognize dates that are expressed in various formats. Valid examples include:

- Today
  - Friday
  - Now
  - 10/30/2020
  - October 30th, 2020
  - October 30th

## Time

A *time* response collects a single time or a range of times. The assistant can recognize times that are expressed in various formats. Valid examples include:

- 12:45PM
  - 10:30
  - 6am
  - Now
  - at 10
  - from 5pm
  - 4 o'clock
  - half past 4

## Currency

A *currency* response collects a currency value, including the amount and the unit. The assistant can recognize currency values that are expressed in various formats. Valid examples include:

- \$10.00
- 20 cents
- five dollars
- 500 yen

## Percent

A *percent* response collects a fractional value that is expressed as a percentage. The assistant can recognize a percentage that is written by using either the percent symbol (%) or the word `percent`). Valid examples include:

- 15%
- 10.5 percent

## Free text

A *free text* response collects any arbitrary text string. Use this response for capturing any text, such as a name or address, or special instructions to be passed along. Valid examples include:

- 123 Main St.
- John Q. Smith
- Please add extra sauce

The default setting for *Free text* is **Always ask for this information** and can't be modified. For more information, see [Skipping steps, always asking steps, or never asking steps](#).

## Saving and reusing customer responses

There might be some questions that your assistant needs to ask in multiple different steps and actions. For example, a banking assistant might support many different actions, each of which requires that the customer specifies an account number. A customer response might have a complex configuration (for example, it might have options with many synonyms). Instead of having to rebuild such a response over and over, you can save a customer response and reuse it wherever your assistant needs it.

### Creating a saved customer response

To create a saved customer response:

1. In **Actions**, click **Saved responses**.
2. Click **New saved response**.
3. In the **Name** field, specify a descriptive name for the saved customer response configuration.
4. Configure the details of the response as required. A saved response can be created only as an options response type. For more information about this response type, see [Customer response types](#).
5. Click **Save**. The saved customer response now appears on the **Saved responses** page.
6. In the **Type of response** field, choose Options or Regex.

 **Important:** You can also edit or delete any existing saved customer response. Keep in mind that any changes that you make apply to all instances of the customer response in any step that uses it. If you delete a saved customer response,

any steps that use that response become invalid and must be corrected to use a different response type.

- Tip:** For the options and regex customer response types, you can also create a saved customer response based on the customer response configuration within a step. If you configure a customer response in a step, click the **Save response for reuse** icon and specify a descriptive name for the saved customer response. (This isn't available if you use the dynamic setting for an options response.)

## Uploading saved customer responses

If you have many saved customer responses, you can upload them from a comma-separated value (CSV) file than to define them one by one. If you are migrating entities from the classic Watson Assistant experience to saved customer responses in the new Watson Assistant experience, see [Migrating intents and entities](#).

1. Collect the saved customer responses into a CSV file. Save the CSV file with UTF-8 encoding and no byte order mark (BOM).

The required format for each line in the file is as follows:

```
$ <savedResponse>,<value>,<synonyms>
```

Where `<savedResponse>` is the name of a saved customer response, `<value>` is a value for the saved customer response, and `<synonyms>` is a comma-separated list of synonyms for that value. For example:

```
$ genres,science fiction,sci-fi,SF
genres,historical fiction,HF
genres,young adult,YA
genres,autobiography
genres,biography
genres,fantasy
locations,Adams Street
locations,Central
locations,South End
```

Uploading a CSV file also supports patterns. Any string that is wrapped with `/` is considered a pattern, as opposed to a synonym. For example:

```
$ ContactInfo,localPhone,/(d{3})-(d{4})/
ContactInfo,fullUSphone,/(d{3})-(d{3})-(d{4})/
ContactInfo,internationalPhone,/^(?/+?[0-9]*\?)?[0-9_\- \(\)]*$/ 
ContactInfo,email,/b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b/
ContactInfo,website,/(https?:\/\/)?([da-z\.-]+)\.([a-z\.]{2,6})([\/\w\.-]*\?)*\?$/
```

2. Go to the **Saved responses** page.
3. Click the **Upload** icon .
4. Select a file from your computer. The maximum CSV file size is 10 MB. If your CSV file is larger, consider splitting it into multiple files and uploading them separately.

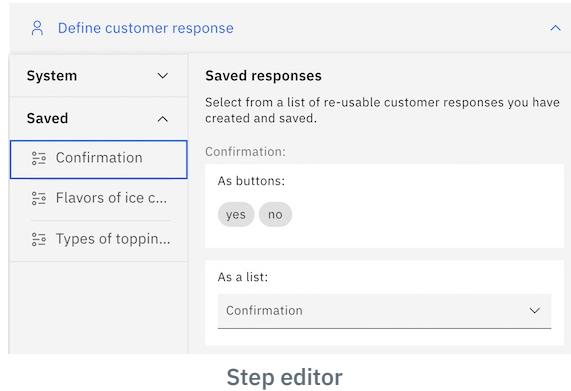
The file is validated and uploaded, and the system trains itself on the new data.

## Using saved customer responses in steps

After you save a customer response, it becomes available as a response type for any step. To use a previously saved customer response in a step:

1. In the step editor, click **Define customer response**.

2. In the list of customer response types, click **Saved** to see the available saved customer responses.



Step editor

3. Click the saved response that you want to use.

To remove a saved response from a step, click the **Delete**  icon. Removing a saved response from a step affects only the step that you are editing. It does not delete the saved response or remove it from any other steps.

To edit a saved response from a step, click **Edit response**. Keep in mind that if you edit a saved response, your changes affect all steps that use the response. If you want to edit the response only for the step you are editing, click the **Unlink from saved response**  icon. After you unlink a response, any edits you make affect only the step you are editing; they do not affect any other steps, nor are they applied to the saved response.

**Tip:** After you unlink a response, you cannot relink it. If you want to return to the saved response without your edits, delete the response, and then read the original saved response. If you want to make your edited version of the response available for reuse, save it as a new saved response.

## Using variables to manage conversation information

When customers reply to your assistant, they share information about themselves and what they want. Your assistant remembers this information, and other information about a conversation, as *variables*. Your assistant can use variables to provide a more personalized and customized experience, and to get users quickly to the solutions they need.

Variables are a powerful tool you can use to build a better assistant. Variables make possible all of the following benefits:

- **Personalization.** The best virtual assistant experiences are targeted and personalized for each customer. When an assistant greets a customer by saying "Hello, Frank! Welcome back," it tells Frank that it remembers his name and that it has talked to him before. By storing this kind of information in variables and then referencing them in your assistant's output, you can personalize the conversation and help your assistant seem more human.
- **Acceleration.** Over the course of a conversation, your customers will answer questions and make choices. These customer responses are stored as variables, which your assistant can then use to guide a conversation. By choosing the right steps and not wasting your customers' time, you can get them as quickly as possible to the right solution.
- **Modularity.** Some information might be useful for many different purposes (for example, a customer's current account balance or contact information). Rather than retrieving or recalculating this information in multiple locations, you can do so once, using a variable to store the result and then access it wherever you need it.

A variable is simply a named container for a piece of information; by referencing this container by name, your assistant can store

or retrieve the information at run time. For example, a variable called *account\_balance* might store your customer's current account balance, a value your assistant can update or retrieve as needed.

The data stored by a variable is characterized by the type of data it contains (such as text, a numeric value, a date, or even a list of multiple values); the operations you can perform with a variable vary depending on its data type.

## Action variables and session variables

Watson Assistant supports two categories of variables:

- **Action variables:** When a step collects information from the customer, the customer response is automatically stored in an *action variable*. You can think of action variables as short-term memory: they persist only for the duration of the current action.

The name of an action variable is always the name of the step that defines the customer response. (You cannot change the name of an action variable.) For example, suppose you define a step that asks "When were you born?" and accepts a date value as a response. The customer response is automatically stored as an action variable called `When were you born?`, which you can then access from any subsequent step in the same action.

- **Session variables:** A value that is not necessarily tied to a particular action can be stored as a *session variable*. Session variables are long-term memory: they persist throughout the user's interaction with the assistant, and your assistant can reference them from any action.

You can create a session variable to store the value from an action variable, if you want to keep the value available for other actions to use. You can also define a session variable based on another session variable, or using a value defined in an expression. In addition to variables you create, Watson Assistant provides a set of built-in session variables for global values like the current time and date.

Session variables can help you to modularize your assistant, because you can write a single action that collects information needed in multiple places. For example, you might have a greeting action that collects basic information about the customer and stores the responses in session variables, which any action can then access.



**Note:** A session variable you create persists only for the duration of a single session. At the end of the session, the variable's value is cleared. How long a session lasts depends upon how your customers access your assistant, and how your assistant is configured.

## Creating a session variable

To add a session variable that can be accessed by any action:

1. From the main actions page, click **Variables > Created by you** in the navigation pane. The list shows all session variables you have created for your assistant.
2. Click **New variable**.



**Tip:** You can also create a new session variable from the step editor. For more information, see [Storing a value in a session variable](#).

3. In the **Name** field, type a name for the session variable.

As you add the name, an ID is generated for you. Any spaces in the name are replaced with underscores (\_) in the ID.

4. **Optional:** Add a type. This sets the response type of the variable. (For more information about response types, see [Choosing a response type](#).)

From this field, you can also select any of the saved responses that you created. For more information about saved responses, see [Saving and reusing customer responses](#).

In addition to the listed types, a variable can also be created as an array. To create an array variable, select **Any** as the type, and in the next step, define an initial value using the expression `[]` to represent an empty array.

5. **Optional:** Add an initial value. This sets the starting value for the variable at the beginning of each user session. For example, suppose you have an assistant your customers can use to make purchases; you might initialize a *Payment due* variable with a starting value of 0, and then add to that value as the customer orders items.

To specify a complex object or an array as the initial value, or to calculate the initial value based on other variables, you can write an expression. For more information about writing expressions, see [Writing expressions](#).

6. **Optional:** Add a description.

7. Click **Apply**.

## Built-in variables

In addition to the variables you create, Watson Assistant provides a set of built-in variables you can access from any action. At run time, these variables are automatically set with the appropriate values. For example, the *Current time* session variable always provides the current time in the user's time zone, at the time of the interaction with the customer.

To see these variables, click **Variables** in the navigation pane from the main actions page.

- The **Set by assistant** page shows built-in session variables that are automatically provided for each assistant.
- The **Set by integration** page shows variables that are automatically provided by the integration your customer is using to connect to the assistant. (These variables are not set if no integration is connected.)

### Set by assistant:

Variable	Variable ID	Description	Examples
<b>name</b>			
<i>Digressed from</i>	<code>digressed_from</code>	Last action before digressing (or null if not digressed)	Pay bill
<i>Now</i>	<code>now</code>	The current date and time in the user's time zone.	2021-08-11T11:28:02
<i>Current time</i>	<code>current_time</code>	The current time in the user's time zone.	11:28:02
<i>Current date</i>	<code>current_date</code>	The current date in the user's time zone.	2021-08-11
<i>Fallback reason</i>	<code>fallback_reason</code>	The reason why a user is routed to the fallback action	Step validation failed - Agent requested - No action matches
<i>No action matches count</i>	<code>no_action_matches_count</code>	This represents a count of customer's consecutive unrecognized input attempts	3

## Variables set by assistant

### Set by integration:

Variable	Variable ID	Description	Example
<b>name</b>			
Timezone	timezone	The user's time zone as specified by the integration or API client. The default time zone (if not specified by the integration) is UTC.	America/New_York
Locale	locale	The user's locale as set by the integration or API client. The locale can affect understanding and formatting of dates, times and numbers.	en-gb
Channel Name	channel_name	The name of the channel that your user is interacting with.	Web_chat

## Variables set by integration

### Storing a value in a session variable

Any action can store a value in a session variable so it is available to other actions. To store a value in a session variable:

1. From within a step, choose the **Set variable values**  icon.
2. Click **Set new value**.
3. In the **Set** drop-down list, select the session or integration variable that you want to store the value in. The new value replaces any previous value that is stored.

If you haven't created the session variable that you want to use, select **New variable**. You can then specify the details about the new session variable, which is added to the list of session variables for the assistant. For more information, see [Creating a session variable](#).

4. In the **To** drop-down list, the choices vary depending on the type of variable you're setting. Possible choices include:

Choice	Description
Scalar value by type	Set a specific value for each variable type. The choice varies depending on the variable type. For example, for a date variable, the choice is <b>Enter a date</b> , and you can use a date picker to set a date. Other choices appear for Boolean, confirmation, currency, date, free text, number, percentage, and time.
Expression	Write an expression to define the value for the session variable. For more information about expressions, see <a href="#">Writing expressions</a> .
Action variables	Select an action variable to use the value of a customer response in another step. The choices that are listed match the type of variable that you want to set.
Session variables	Select another session variable to use its value. The choices that are listed match the type of variable that you want to set.
Assistant variables	Select a built-in system variable to use its value. The choices that are listed match the type of variable that you want to set.
Integration variables	If you are setting an integration variable, you can choose other integration variables as the value.

5. To set more variable values in the same step, click **Set new value**.

## Using variables to manage conversation flow

One of the ways you can use variables is to choose the correct path through the conversation, based on customer responses and other values available at run time. You can do this by defining step conditions, which determine whether a specific step in an action is executed based on runtime conditions.

By defining a condition based on an action variable, you can control whether a step is executed based on the customer's response to a previous step. You can also build step conditions based on session variables, which can store information from other actions.

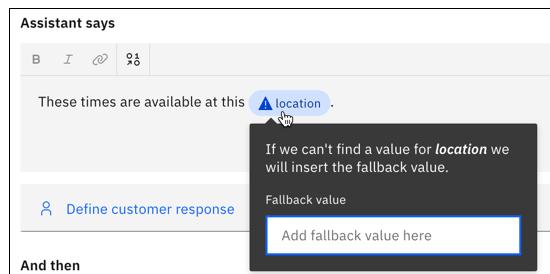
For more information about step conditions, see [Defining step conditions](#).

## Using variables to customize the conversation

You can also use variables in what your assistant says, dynamically referencing information that has been collected during the conversation. This is useful for confirming information the customer has provided (for example, `You want to transfer $153.14 to your checking account. Is that correct?`), and for simply personalizing the conversation to make it more human (`Hi, John. How can I help you today?`).

To reference a variable in what your assistant says:

1. In the **Assistant says** field, start typing the text for the response.
2. When you reach a point where you want to insert a reference to a variable, type a dollar sign (\$) or click the *Insert a variable* icon (  ). A list appears showing the variables you can choose from.
3. Click a variable to add a reference to it in the text.
4. **Optional:** Click the variable in the text to add a fallback value. The fallback value is the value the assistant will use if a user-defined session variable does not contain a value.



When you reference a variable, it appears using a default format in your assistant's response. The format of the variable might differ from the way the value is stored; for example, a date value of `2021-08-11` is formatted as `August 11, 2021` by default.

The default formats are as follows:

Type	Format	Examples
Options	As chosen by the user	Yes No
Number	Numerals only	1000

Date	Mmm DD, YYYY	Jun 30, 2021
Time	H:MM:SS AM	5:15:00 PM
Currency	Number only, no currency symbol	20
Percent	Number only, no percentage symbol	20
Free text	As entered by the user	Please check that the apples aren't bruised

#### Default formats for variables

**Note:** When building an assistant response that includes variables, you concatenate multiple parts (text strings and variables). A single response can consist of no more than 30 concatenated parts (for example, 15 variables along with 15 text strings).

## Referencing expressions

If you need to reference a dynamic value that is calculated using an expression, you must first assign this value to a session variable. (For more information about how to do this, see [Storing a value in a session variable](#).) You can then reference the session variable in the **Assistant says** field.

Note that the `<? . . . ?>` syntax for referencing expressions in assistant output is not supported in actions.

## Adding conditions to a step

An action represents a business process that helps customers answer their questions or solve their problems. Such a process must adapt to different specifics, based on information provided by customers or otherwise available at run time. For example, the steps for withdrawing money from a savings account might be slightly different from the steps for withdrawing for a checking account.

A step condition is a boolean test, based on some runtime value; the step executes only if the test evaluates as true. This test can be applied to any variable, such as an action variable containing the customer response from a previous step. By defining step conditions, you can create multiple pathways through an action based on different possible runtime values.

For more information about variables, see [Using variables to manage conversation information](#).

A basic step condition is expressed in the following form:

`If {variable} {operator} {value}`

where:

- `{variable}` is the name of a variable or an expression.
- `{operator}` is the type of test to apply to the variable value (for example, `is` or `is not`).
- `{value}` is the value to compare to the variable.

For example, a step condition might read:

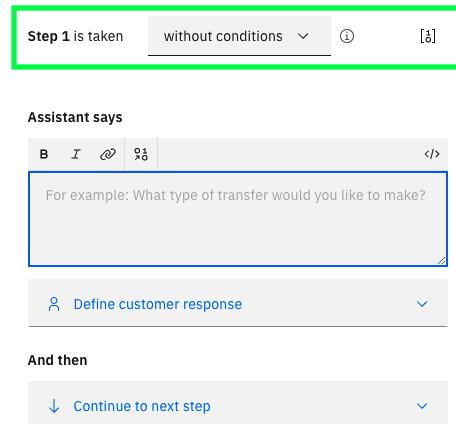
`If Withdraw from which account? is Checking`

This condition evaluates as true if the customer's response to the previous `Withdraw from which account?` step is `Checking`.

Conditions can be grouped together to construct complex tests.

To add a step condition:

1. Open the step. Click the condition field at the beginning of the step:



2. Select **with conditions** from the drop-down list. The **Conditions** section expands.

3. By default, a single condition group, containing a single condition, is automatically created based on the action variable stored by the most recent customer response.

If 1. Withdraw from which account? is Checking x

You can click any part of the expression to edit it:

- o Select the variable you want to test. You can select any of the following:

- An action variable storing the customer response from a previous step in the action
  - A session variable containing a value stored by any action
  - A built-in variable set by the assistant or by an integration

**Note:** You can also define a complex condition by writing an expression defining some other value. For more information about expressions, see [Writing expressions](#).

- o Select the operator representing the test you want to perform on the variable (for example, `is` or `is not`). The available operators for a particular value depend upon its data type. (For more information, see [Operators](#).)
- o Select the value you want to evaluate the test against. Again, the values available depend upon the type of value you are testing. For example, a variable containing an options response can be tested against any of the defined options, and a date value can be tested against any date.

4. To add more than one condition to a step, after adding a condition, click **New condition group**.

One use case where using more than one condition is helpful is if you need to capture a value range. For example, maybe a requirement of opening a checking account is that the customer deposit at least \$100 into the account at creation time. You might ask the customer if they want to transfer funds to the account, and if so, how much? To continue with the transfer, the transfer amount must be \$100 or more, but cannot exceed \$1000. You can add a step with the following conditions:

- How much to transfer? > 99
- How much to transfer? < 1001

Specify whether all or any of the conditions must be met for the step to be included in the conversational flow.

5. To add another group of conditions, click **Add new group**.

You can use groups to build complex step conditions. Each group is evaluated true or false as a whole, and then these results are evaluated together. For example, you might build a step that executes only if all conditions in group 1 are true or any condition in group 2 is true. (Groups function like parentheses in the boolean conditions of many programming languages.)

After you add a group, you can define one or more conditions in the new group. Between groups, choose **and** or **or** to indicate whether the conditions in both conditional groups or only one of them must be met for the step to be included in the conversational flow.

## Operators

An operator specifies the kind of test you are performing on a value in a condition. The specific operators available in a condition depend on the customer response type of the value, as shown in the following table.

Response type	Operators
• Options	<ul style="list-style-type: none"> <li>• is</li> <li>• is not</li> <li>• is any of</li> <li>• is none of</li> </ul>
• Regex	<ul style="list-style-type: none"> <li>• is</li> <li>• is not</li> </ul>
• Number	<ul style="list-style-type: none"> <li>• is defined</li> <li>• is not defined</li> <li>• is equal to (==)</li> <li>• is not equal to (≠)</li> <li>• is less than (&lt;)</li> <li>• is less than or equal to (&lt;=)</li> <li>• is greater than (&gt;)</li> <li>• is greater than or equal to (&gt;=)</li> </ul>
• Currency	
• Percent	
• Date	<ul style="list-style-type: none"> <li>• is defined</li> <li>• is not defined</li> <li>• is on (also allows specific day of the week)</li> <li>• is not on</li> <li>• is before</li> <li>• is after</li> <li>• is on or before</li> <li>• is on or after</li> </ul>
• Time	

- Time
  - is defined
  - is not defined
  - is at
  - is not at
  - is before
  - is after
  - is at or before
  - is at or after

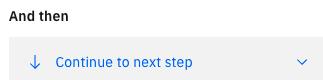
- Free text
  - is
  - is not
  - contains
  - does not contain
  - matches
  - does not match

### Operators

## Choosing what to do at the end of a step

By default, the steps in an action are executed in sequence from first to last. However, you can change this default in order to change what happens next. Used in combination with step conditions, this capability makes it possible for the conversation to follow many different flows depending on the customer's input.

To specify what happens when a step finishes, click **And then**.



Select an option from the drop-down list. The following options are available:

- [Continue to next step](#)
- [Re-ask previous step\(s\)](#)
- [Go to another action](#)
- [Use an extension](#)
- [Search for the answer](#)
- [Connect to agent](#)
- [End the action](#)

### Continue to next step

This option processes the next step in the steps list. As always, the conditions for the next step are evaluated first to determine whether to show the step's response to the customer. This is the default selection.

### Re-ask previous step(s)

This option repeats one or more steps that are listed earlier in the current action. These might be steps that the customer already completed, or steps that were skipped previously based on their step conditions.

You can use this option to handle situations where the customer has made a mistake and asks to go back to a previous point in the conversation. For example, you might include a confirmation step at the end of a process that asks the user whether the collected information is correct; if the user says no, you can return to the beginning of the process. This option is only available

from a step that comes third or later in the steps list.

To repeat previous steps:

1. In the **And then** field, select **Re-ask previous step(s)**.
2. In the Settings window, click to select any previous steps you want to repeat. You can select any step that precedes the step you are editing.

Note that only the selected steps will repeat, regardless of their **And then** settings. Therefore, if you want to repeat the entire action up to this point, you must select all of the previous steps.

**Tip:** The current step you are editing is automatically included in the list of steps to be repeated. To avoid an infinite loop, use step conditions to ensure that this step only executes when it is appropriate for previous steps to be repeated. For example, you might have a step that repeats previous steps only if the user answered **No** to a confirmation question; this way, if the user answers **Yes**, nothing is repeated and the action continues.

3. Click **Apply**.

Any session variable values that were defined based on choices that the customer made in the repeated steps are cleared and replaced with the new responses.

**Note:** There is no option to jump to a later step. Instead of jumping directly to a later step, control the flow through the intervening steps with step conditions or skipping steps.

## Go to another action

This option switches the conversation flow to another action. If you have a portion of an action flow that can be applied across multiple actions (for example, entering credit card information), you can use this capability to build it once and then call to it from each action that needs it. For example, as part of an action to place an order, you might call another action that enables a new customer to create an account.

To call another action

1. In the **And then** field, select **Go to another action**.
2. In the Settings window, click the **Go to** field and select the action you want to call.
3. If you do not want to continue with the current action, click **End this action after the other action is completed**. You might use this option in cases where the customer has decided to do something different; in this case, you want the conversation flow to switch to the other action and not return.

By default, the assistant returns to the current action after the other action completes. Any action variables or session variables defined in the other action are accessible from subsequent steps in the calling action.

4. Click **Apply**.

## Passing values to another action

Optionally, you can pass values to another action so the customer does not need to specify them again. For example, if your order-placement action has already collected the customer's name, you can then pass that information to the account-creation action. The step in the account-creation action that asks for the customer's name is skipped, and the value already specified is used instead.

To pass values to another action:

1. Click **Edit passed values**.
2. In the **Edit passed variables** window, click **Set value for the other action**.
3. Select an action variable that you want to pass a value for. (The available action variables are based on the customer responses defined in the other action.)
4. Select the value you want to pass from the current action. You can select any available variable, or select **Expression** if you want to specify a different value.
5. Click **Apply**.

## Use an extension

You can call an extension that has been added to your assistant in order to interact with an external service. For example, you might use an extension to interact with a ticketing or customer relationship management (CRM) system, or to retrieve real-time data such as mortgage rates or weather conditions. For more information about calling an extension, see [Calling an extension](#).

## Search for the answer

This option indicates that the assistant should use the IBM Watson® Discovery service to search an external data source for information related to the customer's question. To use this option, you must configure a search integration for your assistant. For more information about configuring a search integration, see [Leveraging existing help content](#).

## Connect to agent

This option indicates that the assistant should transfer the conversation to a human agent. For more information, see [Connecting to a live agent](#).

## End the action

This option indicates that this action is complete. Any action variable values that were defined based on choices that the customer made during the action are reset. The action ends immediately, without executing any subsequent steps.

This option can be applied to more than one step in a single action because an action can define more than one branch of a conversation, controlled by step conditions. For example, the open an account action might have one conversational flow for creating a checking account and a separate one for creating a savings account. Each branch might have its own final step. Identifying the final step helps analytical tools that follow a customer's progress through an action to identify the success or failure of the action.

## Adding search

## Administering your instance

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.

<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

## IBM Watson® Discovery search integration setup

PlusEnterprise

The search integration searches for information from a data collection that you create by using the Discovery service.

Discovery is a service that crawls, converts, and normalizes your unstructured data. The product applies data analysis and cognitive intuition to enrich your data such that you can more easily find and retrieve meaningful information from it later. To read more about Discovery, see the [product documentation](#).

**⚠ Important:** The search integration requires Discovery v2. When you create a Plus, Enterprise, or Premium plan instance on IBM Cloud, you get the new and improved version of Discovery. For more information, see [Getting the most from Discovery](#).

Typically, the type of data collection you add to Discovery and access from your assistant contains information that is owned by your company. This proprietary information can include FAQs, sales collateral, technical manuals, or papers written by subject matter experts. Mine this dense collection of proprietary information to find answers to customer questions quickly.

Watch a 4-minute video that provides an overview of the search integration:



**View video:** [Search Integration: Watson Assistant](#)

## Before you begin

1. Before you begin, you must set up a Discovery v2 instance.

You can do this at no cost by using a Plus plan, which offers a 30-day trial. However, to create a Plus plan instance of the service, you must have a paid account (where you provide credit card details).

2. Create a Plus plan Discovery service instance.

Go to the [Discovery](#) page in the IBM Cloud catalog and create a Plus plan service instance.

 **Important:** If you decide not to continue using the Plus plan and don't want to pay for it, delete the Plus plan service instance before the 30-day trial period ends.

## Create the search integration

1. From the assistant where you want to add search, click **Integrations**.

 **Note:** You can add search if you are a user with a paid plan.

2. In the **Extensions** section, locate Search, click **Add**, then click **Confirm**.

3. Next, connect to a Discovery service instance.

## Connect to an existing Discovery instance

1. Choose the Discovery service instance that you want to extract information from.

 **Note:** If you see a warning that some of your Discovery service instances do not have credentials set, it means that you can access at least one instance that you never opened from the IBM Cloud dashboard. You must access a service instance for credentials to be created for it, and credentials must exist before Watson Assistant can establish a connection to the Discovery service instance on your behalf. If you think a Discovery service instance is missing from the list, open the instance from the IBM Cloud® dashboard directly to generate credentials for it.

2. Indicate the data collection to use, by doing one of the following things:

- Choose an existing project.

You can click the *Open Discovery* icon to review the configuration of a project before you decide which one to use.

Go to [Configure the search](#).

- If you do not have a project or do not want to use any of the projects that are listed, click **Create a new project** to add one. Follow the steps in [Create a project](#).

 **Note: Create a new project** is not displayed if you reached the limits based on your Discovery service plan. See [Discovery pricing plans](#) for plan limit details.

## Create a project

1. On the **OK, where is your data?** page, select your data source, then click **Next**. Example choices include Salesforce, SharePoint, Box, IBM Cloud Object Storage, web crawl, and upload data.
2. On the **Let's create a collection for your data** pages, enter the information on how to connect to your data and configure

your collection. The information that you need to enter is different depending on the data source. For example, you need to enter your authentication credentials for services such as Salesforce, Sharepoint, and Box. For web crawl, you specify the website with your existing information.

3. Click **Finish**. Give Discovery a few minutes to start creating documents. You can use the **Manage collections** page within the project to see the progress.
4. Wait for the collection to be fully ingested, then click **Back to Watson Assistant**.

## Configure the search

1. On the Watson Assistant search integration page, verify that the Discovery instance and project you want to use is selected, then click **Next**.
2. In the **Configure result content** section, review the Discovery fields and examples that are used in the search results shown to your customers. You can accept the defaults, or customize them as you want.

The appropriate collection fields to extract data from vary depending on your collection's data source and how the data source was enriched. After you choose a data collection type, the collection field values are prepopulated with source fields that are considered most likely to contain useful information given the collection's data source type. However, you know your data better than anyone. You can change the source fields to ones that contain the best information to meet your needs.

To learn more about the structure of the documents in your collection, including the names of fields that contain information you might want to extract, open the collection in Discovery, and then use the **Identity fields** and **Manage fields** tabs.

Each search result can consist of the following sections:

- **Title:** Search result title. Use the title, name, or similar type of field from the collection as the search result title. You must select something for the title or no search result response is displayed in the Facebook and Slack integrations.
- **Body:** Search result description. Use an abstract, summary, or highlight field from the collection as the search result body. You must select something for the body or no search result response is displayed in the Facebook and Slack integrations.
- **URL:** This field can be populated with any footer content that you want to include at the end of the search result.

For example, you might want to include a hypertext link to the original data object in its data source. Most online data sources provide self-referencing public URLs for objects in the store to support direct access. If you add a URL, it must be valid and reachable. If it is not, the Slack integration doesn't include the URL in its response and the Facebook integration doesn't return any response.

The Facebook and Slack integrations can successfully display the search result response when the URL field is empty.

**⚠Important:** You must use a field for at least one of the search results.

If no options are available from the drop-down fields, give Discovery more time to finish creating the collection. If the collection is not created, then your collection might not contain any documents or might have ingestion errors that you need to address first.

As you add field mappings, a preview of the search result is displayed with information from the corresponding fields of your data collection. This preview shows you what gets included in the search result response that is returned to users.

To get help with configuring the search, see [Troubleshooting](#).

3. Use the **Message**, **No results found** and **Connectivity issue** tabs to customize different messages to share with users based on the successfulness of the search.

Tab	Scenario	Example message
Message	Search results are returned	I found this information that might be helpful:
No results found	No search results are found	I searched my knowledge base for information that might address your query, but did not find anything useful to share.
Connectivity issue	I was unable to complete the search for some reason	I might have information that could help address your query, but am unable to search my knowledge base at the moment.

#### Search result messages

4. Choose whether to enable **Emphasize the answer**.

**Note:** This option is available only if your Discovery instance uses the v2 Discovery API.

When you enable this feature, the sentence that is determined by Discovery to be the exact answer to the customer's question is highlighted in the block of text that is displayed to the customer as the search result.

5. In the **Adjust result quantity** section, specify the number of results to return.

The top three results are returned automatically. You can choose to show fewer or more (up to 10) results in the response.

By default, customers can choose to see more results. If you don't want to give customers this choice, clear the **Include link for customers to view up to 10 results** checkbox.

6. In the **Set result selectivity** section, decide whether to be more selective with the answers that are returned. By increasing result selectivity, Search returns fewer but more accurate results. In most cases, Search is accurate enough that the default setting (off) is sufficient.

7. Click **Preview**. Enter a test message to see the results that are returned when your configuration choices are applied to the search. Make adjustments as necessary.
8. Click **Create**.

## Edit the search integration configuration

If you want to change the configuration of the search result card later, open the search integration again, and make edits. You do not need to save changes as you make them; they are automatically applied. When you are happy with the search results, click **Save** to finish configuring the search integration.

If you decide you want to connect to a different Discovery service instance or project, open the search integration and click **Edit Discovery Settings**. You can choose either a new project from the same instance, or a new instance and project.

## Troubleshooting

Review this information for help with common tasks.

- **Creating a web crawl data collection:** Things to know when you create a web crawl data source:
  - To increase the number of documents that are available to the data collection, click add a URL group where you can list the URLs for pages that you want to crawl but that are not linked to from the initial seed URL.
  - To decrease the number of documents that are available to the data collection, specify a subdomain of the base URL. Or, in the web crawl settings, limit the number of hops that Watson can make from the original page. You can specify subdomains to explicitly exclude from the crawl also.
  - If no documents are listed after a few minutes and a page refresh, then make sure that the content you want to ingest is available from the URL's page source. Some web page content is dynamically generated and therefore cannot be crawled.
- **Configuring search results for uploaded documents:** If you are using a collection of uploaded documents and cannot get the correct search results or the results are not concise enough, consider using *Smart Document Understanding* when you create the data collection.

You can annotate documents based on text formatting. For example, you can teach Discovery that any text in 28-point bold font is a document title. If you apply this information to the collection when you ingest it, you can later use the *title* field as the source for the title section of your search result.

You can also use Smart Document Understanding to split up large documents into segments that are easier to search. For more information, see the [Smart Document Understanding](#) topic in the Discovery documentation.

- **My response text is surrounded by brackets:** If you notice that your response text is surrounded by brackets and quotation marks ( `["My response text"]` ) when you test it from the Preview, for example, you might need to change the source field that you're using in the configuration. The unexpected formatting indicates that the value is stored in the source document as an array. Any field that you extract text from must contain a value with a String data type, not an Array data type. When the chat integration shows a response that is extracted from a field that stores the data as an array, it does a straight conversion of the array value into a string, which produces a response that includes the array syntax.

For example, maybe the field in the source document contains an array with a single text value as its only array element:

```
"title": ["a single array element"]
```

The array value is converted by the Watson Assistant into this string value:

```
"title": "[\"a single array element\"]"
```

As a result, the string is returned in this format in the chat; the surrounding brackets and quotation marks are displayed:

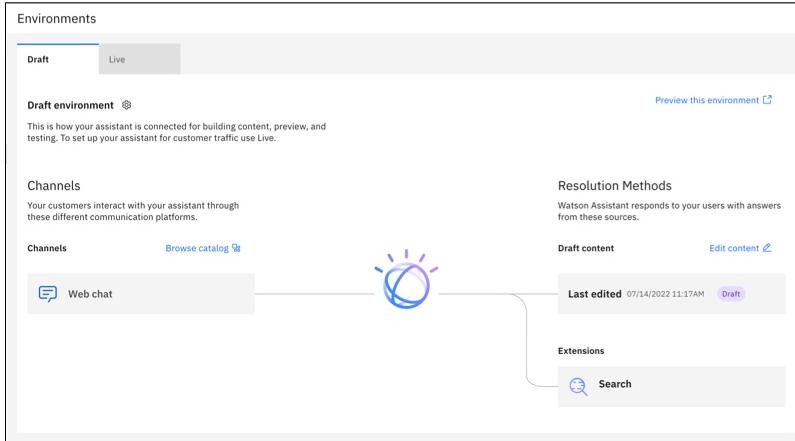
```
[ "a single array element"]
```

If you see this happening, consider choosing a different collection field from which to extract search results.

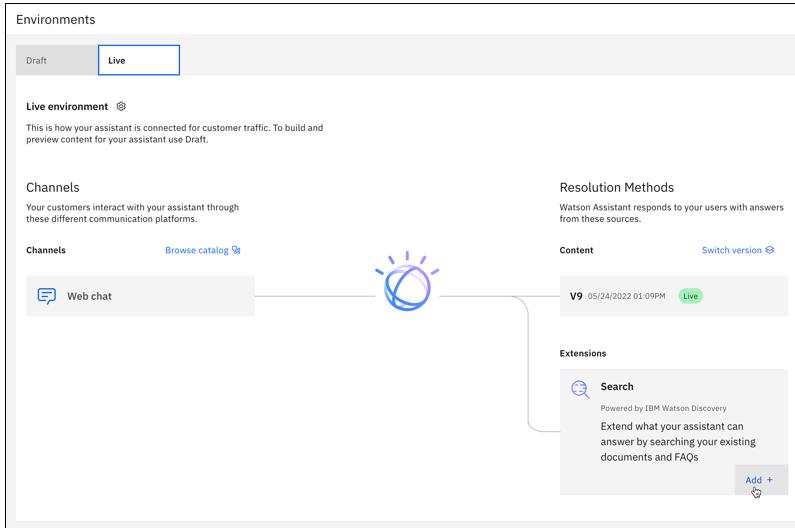
**Note:** The Discovery document `highlight` field stores values in an array.

## Next steps

After you add the search integration for the first time, it appears as a tile on the **Draft environment** page. Click the tile to see or edit the search configuration.



When you're ready, you can repeat the steps to add the search integration to your **Live environment** or to other environments, if you're using [multiple environments](#).



## Search trigger

The search integration is triggered from an action step. This approach is useful if you want to narrow down a user query before you trigger a search.

For example, the conversational flow might collect information about the type of device a customer wants to buy. When you know the device model, you can then send a model keyword in the query that is submitted to the search integration to get better results.

In the *And then* field of the step where you want the search to be triggered, choose **Search for the answer**.

To configure the search in Discovery, complete the following steps:

1. Click **Edit settings**.

2. Add values to one or both of the following fields:

- **Custom search query**: Add a word or phrase that you want to submit to Discovery as the query string for the search.

For example, you can specify a string such as, `What cities do you fly to?`.

For a more dynamic string, you can include a variable. For example, `Do you have flights to ${destination}?`

You are effectively defining the value that is used by the Discovery API as the `natural_language_query` parameter.

For more information, see [Query parameters](#).

If you don't specify a text string, the action sends the most recently submitted user message as the search string.

If you want to use the original customer message that triggered the action as the query string instead, you need to plan ahead. You can follow these steps:

1. Create a session variable to store the initial user input. For example, named `original_message`.
2. In Step 1, meaning the first step after the action trigger, set the value of the session variable. For more information about session variables, see [Defining session variables](#).
3. Set the value of the variable by using an expression that looks like this: `<? input.text ?>`.

This expression captures the complete message that was submitted by the customer. As a result, your variable captures the customer message that triggered this action.

1. Add the session variable to the *Custom query* field (for example,  `${original_message}`).
- **Custom results filter**: Add a text string that defines information that must be present in any of the search results that are returned.

You are effectively defining the value that is used by the Discovery API as the `filter` parameter. For more information, see [Query parameters](#).

The syntax to use for the filter value is not intuitive. Here are a few examples of common use cases:

- To indicate that you want to return only documents with positive sentiment, for example, specify `enriched_text.sentiment.document.label:positive`.
- To filter results to include only documents that mention `Boston, MA`, specify `enriched_text.entities.text:"Boston, MA"`.
- To filter results to include only documents that mention a city name that you saved in a context variable named `$destination`, you can specify `enriched_text.entities.text:$destination`.

If you add both a query and a filter value, the filter parameter is applied first to filter the data collection documents and cache the results. The query parameter then ranks the cached results.

3. If you want the search for an answer to be the last step in the action, select **End the action after returning results**.
4. Click **Apply**.

## Test the search integration

After you configure search, you can send test queries to see the search results that get returned from Discovery by using the Preview page.

To test the full experience that customers have when they ask questions that are either answered by the action or trigger a search, use the *Preview* for your assistant.

## Use search when no action matches

You can use the search integration with the built-in [No action matches](#) capability. By adding search to *No action matches*, you can have your assistant refer to search when a customer asks a question that isn't addressed by an existing action.

To update *No action matches* to use search:

1. In your assistant, click **Actions**, then click **Set by assistant**.
2. Click **No action matches** to open it in the editor.
3. Click **New step**.
4. In the **And then** section, click **Continue to next step**, then choose **Search for the answer**.
5. Because you're adding search, you no longer need step 2, which is the step for when the **No action matches** count is 3 or less. Click the delete (trash can) icon to remove it.
6. Close **No action matches**. Now your assistant uses search to provide customers with potentially useful answers, if the customer question does not trigger any of the existing actions.

## Coveo search extension setup

You can use [Coveo](#) to create and manage source documents. The [Coveo Search API](#) can issue search queries on an index. It is a configurable search that you can customize based on your use case. You can access Coveo search through an extension to your assistant.

To set up the extension for Coveo search:

### Add an API key

In the Coveo Administration Console, add an API key. For detailed instructions, see [Add an API Key](#) in the Coveo documentation. Ensure that the API key is set to enable search and to allow `Execute queries`.

### Download the OpenAPI specification

Download the OpenAPI specification file: [coveo.openapi.json](#). You use this file to add the extension to your assistant.

The OpenAPI specification defines the following method:

- `GET /rest/search/v2`: Search for content in a set of sources or documents.

For more information about the endpoints, see [Perform a Query](#) in the Coveo documentation.

### Create and add extension

1. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see [Building the custom extension](#).
2. After you build the Coveo extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your Coveo API key to authenticate. For general instructions on adding any custom extension, see [Adding an extension to](#)

[your assistant](#).

## Add and edit the Coveo search starter kit action template

1. If you haven't already, use **Quick start with templates** to add the Coveo search starter kit. The starter kit adds an action for use with Coveo search. For more information, see [Building actions with templates](#).

 **Note:** **Quick start with templates** is available in English-language assistants only.

2. On the **Actions** page, edit the **\*Coveo search** action to use the extension. In step 3, click **Edit extension**.
3. In the **Extension** field, choose the Coveo extension that you built.
4. In the **Operation** field, choose `Search request to Coveo search`.
5. In the **Parameters** list, set `q` to `*query_text`.
6. Close the **Coveo search** action.

## Edit system actions

1. Click **Set by assistant** and open the **No action matches** action.
2. Delete the two default steps.
3. Add a step. Set **And then** to **Go to another action** and choose the **Coveo search** action.
4. If you aren't connecting your customers to a live agent, you might want to edit the **Fallback** action in the same way as **No action matches**.

## Using your Coveo search extension

Issue a query to your assistant. If no action that matches that query, then it uses Coveo to produce search results.

## Google custom search extension setup

You can access Google search through an extension to your assistant that uses the [Google Programmable Search Engine](#). It is a configurable search that you can customize based on your use case.

To set up the extension for Google search:

### Get Search Engine ID and API key

Create a Google Programmable Search Engine. Then, get its Search Engine ID and an API key. For detailed instructions, see [Create Programmable Search Engine](#) in the Google Programmable Search Engine documentation.

### Download the OpenAPI specification

Download the OpenAPI specification file: [google-custom-search-openapi.json](#). You use this file to add the extension to your assistant.

The OpenAPI specification defines the following methods:

- `GET /customsearch/v1`: Search for content over the entire web.
- `GET /customsearch/v1/siterestrict`: Search for content over a specific collection of websites.

For more information about the endpoints, see [Custom Search](#) or [Custom Search Site Restricted](#).

The endpoints have the same arguments and responses, but with differences:

- *Custom Search Site Restricted* is restricted to searching 10 or fewer websites, each of which can have an unlimited number of pages.
- *Custom Search* can support any number of websites that are indexed by Google, but has a [daily query limit](#).

For a typical assistant focused on a specific topic, it is usually only necessary to search a single website or a few websites.

*Custom Search Site Restricted* is a better fit since it doesn't have a limit on the number of queries that can be run per day.

Assistants that need to search more than 10 websites need to use *Custom Search* instead.

## Create and add extension

1. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see [Building the custom extension](#).
2. After you build the Google custom search extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your Google programmable search engine API key to authenticate. For general instructions on adding any custom extension, see [Adding an extension to your assistant](#).

## Add and edit the Google custom search starter kit action template

1. If you haven't already, use **Quick start with templates** to add the Google custom search starter kit. The starter kit adds an action for use with Google custom search. For more information, see [Building actions with templates](#).

 **Note:** **Quick start with templates** is available in English-language assistants only.

2. On the **Actions** page, edit the **\*Google search** action to use the extension. In step 3, click **Edit extension**.

3. In the **Extension** field, choose the Google custom search extension that you built.

4. In the **Operation** field, choose either **Custom Search** or **Custom Search Site Restricted**.

This table compares the operations:

Operation	Restrictions	Daily query limit
Custom Search	Supports any number of websites that are indexed by Google	Yes
Custom Search Site Restricted	Restricted to searching 10 or fewer websites, each of which can have an unlimited number of pages	No

### Operations

5. In the **Parameters** list, set:

- `q` to `*query_text`
- `cx` to `*cx`

6. Click **Optional parameters** to expand the list. Set `num` to `*num_of_results`.

7. Close the **Google search** action.

## Edit system actions

1. Click **Set by assistant** and open the **No action matches** action.
2. Delete the two default steps.
3. Add a step. Set **And then** to **Go to another action** and choose the **Google search** action.
4. If you aren't connecting your customers to a live agent, you might want to edit the **Fallback** action in the same way as **No action matches**.

## Using your Google custom search extension

Issue a query to your assistant. If no action that matches that query, then it uses Google to produce search results.

### Limit on search results size

Watson Assistant has a 100 kb limit on the size of information that is stored in context variables, which includes search results. If the results from your extension exceed that limit, the action can fail without any visible warning or error. Typically a long delay occurs and then there is no response. This failure rarely happens with the Google custom search extension, but it might happen if you are searching a site with large volumes of metadata that is returned by Google custom search. If you think that this might be a problem, try running the query in an API testing tool like curl, [Insomnia](#), or [Postman](#). Check how many bytes of data you are getting as search results. If the total is at or near 100 kb, you might be able to work around the issue by reducing `num_of_results` and getting fewer results for each query or by excluding sites or pages with large volumes of metadata.

For more information, see [Limit on Size of Search Results](#) in a starter kit for IBM Watson® Discovery.

## NeuralSeek extension setup

[NeuralSeek](#) by [Cerebral Blue](#) is a combined search and natural-language generation system that is designed to [make conversational AI feel more conversational](#). It requires that you load all your content into [IBM Watson® Discovery](#). Then, when a user asks a question, it has Discovery search for multiple relevant documents and then it generates a natural-language answer that uses the contents of those documents. In some cases, the answer might be taken directly from a single document, and in others, the answer can include information from multiple sources that are combined into a single coherent statement. For each query, NeuralSeek returns a single answer and a confidence score. In most cases, it also returns a URL of a document that influenced the answer, which might be one of several documents.

To set up the extension for NeuralSeek search:

### Set up IBM Watson® Discovery

1. You need an instance of [IBM Watson® Discovery](#). Because NeuralSeek can modify your data as needed to make it more effective, make sure it is not an instance with important data that you are using for other purposes.
2. In Discovery, create a project and load the documents that you want to use.

### Get the NeuralSeek OpenAPI specification and API Key

1. You also need an instance of [NeuralSeek on IBM Cloud](#).
2. In NeuralSeek, open the **Configure** page and enter your information in the **Discovery instance details** section.
3. On the **Integrate** page, and click the **OpenAPI file** link to download the `NeuralSeek.json` OpenAPI specification file configured for your instance.
4. Also on the **Integrate** page, copy the API key for NeuralSeek. The API key for NeuralSeek is not the same as the API key for Discovery that you used on the **Configure** page. The API key for NeuralSeek is only available on the **Integrate** page.

 **Note:** Simple instructions for setting up NeuralSeek are available on the [Integrate](#) page. You can also follow those instructions to get NeuralSeek working with Watson Assistant.

## Create and add extension

1. In your assistant, on the [Integrations](#) page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see [Building the custom extension](#).
2. After you build the NeuralSeek extension and it appears on your [Integrations](#) page, click **Add** to add it to your assistant. Use your NeuralSeek API key to authenticate. For general instructions on adding any custom extension, see [Adding an extension to your assistant](#).

## Add and edit the NeuralSeek starter kit action template

1. If you haven't already, use [Quick start with templates](#) to add the NeuralSeek starter kit. The starter kit adds an action for use with NeuralSeek. For more information, see [Building actions with templates](#).

 **Note:** [Quick start with templates](#) is available in English-language assistants only.

2. On the [Actions](#) page, edit the **\*NeuralSeek search** action to use the extension. In step 3, click **Edit extension**.
3. In the **Extension** field, choose the NeuralSeek extension that you built.
4. In the **Operation** field, choose `Seek an answer from NeuralSeek`.
5. In the **Parameters** list, set `question` to `*query_text`.
6. Click **Optional parameters** to expand the list. Set these parameters:
  - Set `context` to `*query_context`.
  - You might want to also set the `language` parameter to an expression that contains a 2-letter language code such as `en`.
7. Close the **\*NeuralSeek search** action.

## Edit system actions

1. Click **Set by assistant** and open the **No action matches** action.
2. Delete the two default steps.
3. Add a step. Set **And then** to **Go to another action** and choose the **NeuralSeek search** action.
4. If you aren't connecting your customers to a live agent, you might want to edit the **Fallback** action in the same way as **No action matches**.

## Using your NeuralSeek extension

Issue a query to your assistant. If no action that matches that query, then it uses NeuralSeek to produce search results.

For many use cases, NeuralSeek alone is enough to deploy an assistant. If you are happy with your assistant, you might want to deploy it for real-world use right away. Use the [Analyze](#) page in Watson Assistant or the [Curate](#) page in NeuralSeek to see what kinds of questions users are asking and actions for the common user requests. The [Curate](#) page can automate the creation of new actions and generate new example utterances that trigger those actions. It can also merge any existing actions with the

actions that it generates so you can update an existing assistant. For more information, see the [NeuralSeek documentation](#)

## Calling a custom extension

An extension is an integration with an external service. By calling an extension from an action, your assistant can send requests to the external service and receive response data it can use in the conversation.

For example, you might use an extension to interact with a ticketing or customer relationship management (CRM) system, or to retrieve real-time data such as mortgage rates or weather conditions. Response data from the extension is then available as action variables, which your assistant can use in the conversation.

For information about how to build a custom extension, see [Build a custom extension](#).

## Calling the extension from a step

To call a custom extension from an action:

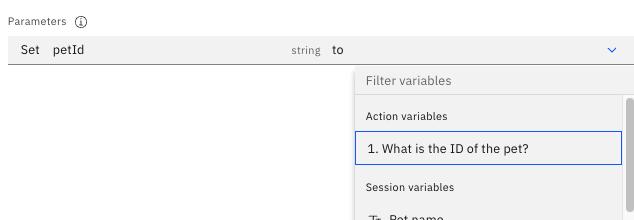
1. In the action editor, create or open the step from which you want to call the extension.
2. **Optional:** In the **Assistant says** field, type a message to be shown to the customer before the extension is called (for example, `Please wait while I retrieve your account balance...`).

The output from this step is sent to the channel with the global context variable `skip_user_input` set to `true`. This variable tells the channel to display the message but *not* to prompt the customer for a reply. Instead, the channel sends an empty message, enabling the assistant to proceed with the call to the extension.

**Note:** All built-in channel integrations (such as the web chat) respect the `skip_user_input` context variable. If you are using the API to develop a custom client, it is your responsibility to include logic checking for this variable. For more information, see [Processing user input](#).

3. In the step editor, click **And then**.
4. Click **Use an extension**.
5. In the **Extension setup** window, specify the following information:
  - In the **Extension** field, select the extension you want to call.
  - In the **Operation** field, select the operation you want to perform. (An *operation* is a method or function supported by the extension.)
6. Specify values for each of the required input parameters. A *parameter* is an input value sent to an operation, such as the ID of a customer record you want to retrieve or the location to use for a weather forecast.

To assign a value to a parameter, click the input field for the value. You can then select from the list of available variables or write an expression to specify the value.

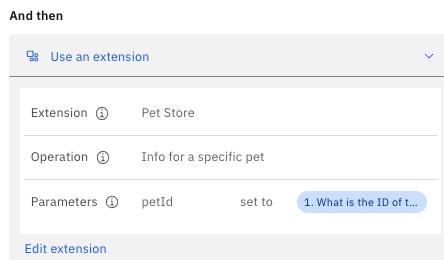


Each parameter has a data type (such as *number* or *string*). The variable you select must be compatible with the data type of the parameter; for more information, see [Compatible variables for parameters](#).

You must specify values for all required parameters before you can proceed.

7. If you want to specify a value for any optional parameters, click **Optional parameters**. You can then repeat this process for each optional parameter you want to use.
8. Click **Apply**. (If the **Apply** button is not available, check to make sure you have specified values for all required parameters.)

The **And then** section of the step editor now shows an overview of the call to the extension:



If you need to make changes, click **Edit extension** to reopen the **Extension setup** window.

## Compatible variables for parameters

To pass an input parameter value for an operation, you must select a compatible action variable or session variable.

An action variable contains a value that is based on a customer response in a previous step. A session variable might have a value based on a customer response or a value defined by an expression. (For more information about action variables and session variables, see [Using variables to manage conversation information](#).)

When you assign a value to a parameter, the variable you choose must be compatible with the data type of the parameter. (For example, a *number* parameter must be assigned a numeric value rather than text.)

The following table shows the possible customer response types and the parameter data type compatible with each.

Customer response type	Compatible data types	Notes
options	string	A selected option is always treated as a string, even if it is a numeric value.
number	number integer	A floating-point number passed as the value for an <code>integer</code> parameter might cause an error, depending on the behavior of the REST API.
date	string	Dates are rendered as <code>YYYY-MM-DD</code> .
time	string	Times are rendered as <code>HH:MM:SS</code> in 24-hour format, converted to the user's time zone.
Currency	number integer	
Percent	number integer	A percent value is passed as an integer (so <code>75%</code> becomes <code>75</code> ).

Free text	string
Regex	string

#### Compatible response types for parameters

## Arrays

In addition to the supported customer response types, a variable can also contain an array value. If you need to pass an array parameter to an operation, you must create an array session variable:

1. Create a new session variable, either using the **Set variable values** icon in the step editor or from the **Variables > Created by you** page. (For more information about how to create a session variable, see [Creating a session variable](#).)
2. In the **Type** field, select **Any**.
3. In the **Initial value** field, click the **Use expression** toggle to enable it. Enter an expression that defines an array value (such as `["New York", "London", "Tokyo"]`, `[123, 456, 789]`, or `[]`).



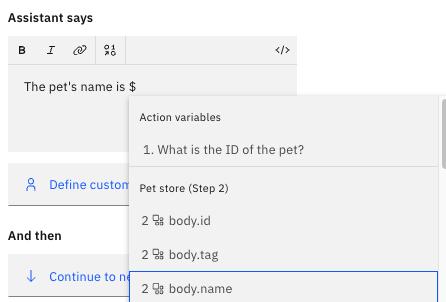
**Note:** Because this variable contains an array value, your actions can use expressions with array methods to access or modify the array values. For example, you might want to create a variable that initially contains an empty array (`[]`) and then use the `add()` method to build a list one element at a time. For more information about the array methods you can use in expressions, see [Array methods](#).

You can now select this variable as the value for a parameter that requires an array.

## Accessing extension response data

After you call an extension, values from the response data are stored in special action variables that you can access in subsequent steps.

You can access these variables in the same way you access other action variables. You can reference it in the **Assistant says** text, evaluate it as part of a step condition, or assign it to a session variable so other actions can access it. The response variables are shown in the list of available variables, categorized under the name of the extension and the step from which it was called:



Each call to an extension creates a separate set of response variables. If your action calls the same extension multiple times from different steps, make sure you select the variables from the correct step.

Each variable represents a value from the response body. To make it easy to access these values, data is extracted from complex, nested objects and mapped to individual response variables. The name of each variable reflects its location within the response body (for example, `body.name` or `body.customer.address.zipcode`).

For example, this action step uses an expression to check the `availability` property in an extension response:

Conditions 1 condition 

1. All  of this is true:

If  `body.availability == "unavailable"` 

and [Add condition +](#)

---

2. [New group +](#)

---

**Assistant says**

I'm sorry, that pet is no longer available.

If a response variable contains an array, you can write an expression that uses array methods to access the elements of the array. For example, you might use the `contains()` method in a step condition to test whether the array contains a particular value, or the `join()` method to format data from the array as a string you can include in an assistant response. For more information about array methods, see [Array methods](#).

## Checking success or failure

You might want your assistant to be able to handle errors that occur when calling a custom extension. You can do this by checking the `Ran successfully` response variable that is returned along with the response from the call to the extension. This variable is a boolean (`true` or `false`) value.

If you define step conditions that check the `Ran successfully` variable, you can create steps that enable your assistant to respond differently depending on whether the call to the extension succeeded. (For more information about step conditions, see [Step conditions](#).)

The following example shows a step condition that checks for a failure from an extension in step 3. By using this condition, you can create a step that tells the customer there was an error, and perhaps offers to connect to an agent for more help.

## Conditioning on HTTP status

In addition to the `Ran successfully` variable, you might also want to create a step condition based on the HTTP status of the response. By doing this, you can create steps that handle the situation differently depending on the cause of the failure. For example, if the call failed because of a timeout error (HTTP status 408), you might want to retry the call.

**⚠ Important:** There are many possible HTTP status codes, and different methods use different status codes to indicate various types of success or failure. To condition on the HTTP status, you need to know what HTTP status codes the external service returns, and under what circumstances. These status codes are typically specified in the OpenAPI document that describes the external API.

To create an step condition based on the HTTP status code, follow these steps:

1. For the value you want to test, click **Expression**.
2. In the expression field, type a dollar sign (\$) to show the list of available variables.
3. Select any variable that is a response value from the extension. (It doesn't matter which variable you select, as long as it is an extension response variable).

The expression is automatically updated to show a reference to the variable you selected, in the format

`${step_xxx_result_y.body.variablename}`. For example, if you selected a response variable called `body.id`, the reference might be  `${step_596_result_1.body.id}`.

4. Inside the curly braces, ({ }), edit this reference to remove `.body.variablename`. You should be left with something like  `${step_596_result_1}`.
5. After the closing curly brace (}), add `.status`. The resulting reference identifies the status code returned from the call to the extension (for example,  `${step_596_result_1}.status`).

For more information about writing expressions, see [Writing expressions](#).

6. Complete the expression by adding an operator and a comparison value, so the expression evaluates to a Boolean (true/false) value. For example, the following expression tests for HTTP status 408, which indicates a timeout error:

```
$ ${step_549_result_1}.status==408
```

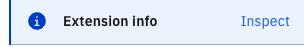
## Debugging failures

If your calls to an extension are failing, you might want to debug the problem by seeing detailed information about what is being sent to and returned from the external API. To do this, you can use the extension inspector in the Preview pane:

1. On the Actions page, or in the action editor, click **Preview** to open the Preview pane.

**Note:** You cannot access the extension inspector from the assistant preview on the **Preview** page, which shows only what a customer would see. Instead, use the preview feature that is part of the Actions page, which gives you access to additional information.

2. Interact with your assistant as a customer would.
3. Each time an extension is called, the preview pane shows a message giving you access to detailed information:



Click **Inspect** to see details about the call to the extension.

**Tip:** You can also click the  icon to show or hide the extension inspector. However, you must click **Inspect** in the preview pane to show information about a particular call to an extension.

The **Overview** tab of the extension inspector shows the following information about a call to an extension:

### Extension

The name of the extension, as specified in the extension settings.

### Operation

The operation that was called.

#### Status

The HTTP status code from the response. This code can help you determine if an error is being returned from the external service.

#### Request parameters

The input parameters that were sent to the external API as part of the request.

#### Response properties

The values of all properties included in the response from the external API. These are the values that are mapped to action variables after the call to the extension completes.

 **Tip:** In the **Request parameters** and **Response properties** tables, long property names might be truncated to show only the last part of the JSON path. To see the complete path and property name, hover the mouse pointer over the property name in the table.

4. Click the **Advanced** tab in the extension inspector if you want to see the raw request and response data:

- The request is shown as a cURL command, which you can run at a command prompt or import into a tool such as [Postman](#). (For security reasons, the content of any Authorization header is not included.)
- The response is shown as the complete JSON data returned from the external API.

## Reconfiguring a missing extension

An extension might become unavailable if someone removes it from the assistant on the **Integrations** page, or if the action is exported and then imported to a different assistant where the required extension is not configured. If this happens, any action step that calls the extension becomes invalid.

To correct the problem, follow these steps:

1. If necessary, recreate the extension using the same OpenAPI specification that was used before. (For more information, see [Building a custom extension](#).)
2. Make sure the extension has been added to the assistant. (For more information, see [Adding an extension to your assistant](#).)
3. In the action editor, edit the action step that calls the extension and check whether the call to the extension is correctly configured. If Watson Assistant recognizes the required extension, the extension configuration is automatically restored.

If you see the message `Extension not fully configured`, this means that Watson Assistant did not find the required extension. Click **Edit extension**.

4. In the **Extension setup** window, select the extension you want to call.

 **Note:** If Watson Assistant recognizes an available extension that was built using the same OpenAPI document, a message appears suggesting that you select this extension. However, you can select any available extension.

5. Verify that the correct values are specified in the **Operation** and **Parameters** fields.

6. Click **Apply**.

7. If you selected an extension that is not identical to the one that was used to build the action, you might need to modify

subsequent steps that access the extension response properties. Check any later steps that refer to the response properties and make sure the references are still valid and correct.

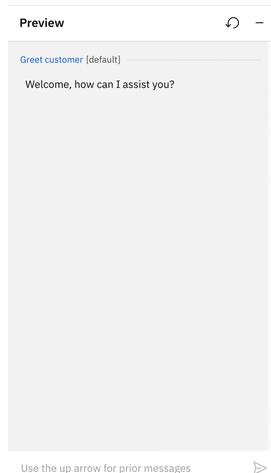
## Allowing your customers to change the topic of the conversation

In general, an action is designed to lead a customer through a particular process without any interruptions. However, real conversations almost never follow such a simple flow. In the middle of a conversation, customers might get distracted, ask questions about related issues, misunderstand something, or change their minds about what they want to do.

The **Change conversation topic** feature enables your assistant to handle these digressions, dynamically responding to the user by changing the conversation topic as needed.

### How changing the topic works

This example shows a customer who changes the conversation topic while they are entering a credit card. When the assistant asks `What is your CVV number?`, the customer (who is new to credit cards) asks what a CVV is. The assistant is designed to handle this possibility, so it switches to a different action that answers the customer's question. It then continues where it left off.



The assistant determines when to change the conversation topic as follows:

1. When the assistant asks a question and receives a response, it first validates the response to see whether it answers the question. In the CVV example, the assistant expects a number as a response.
2. If the input is not recognized as an answer to the question, the assistant then evaluates the input to see whether it matches another action. (Changing the topic cannot switch the conversation to a different assistant.)

**Note:** If the input does not answer the question, and it does not match any existing action of the assistant, a step validation error results. For more information about validation errors and how they are handled, see [Handling errors in the conversation](#).

3. If the input matches a different action, the assistant switches to the matching action. The assistant changes to the new action if it has a confidence score match of 20% or higher. In the example, the customer's response (`What's a CVV number?`) is not a valid response, but it does match another action that is designed to answer this question. The matching action is triggered, answering the customer's question.
4. After the second action completes, the assistant asks the customer if they want to return to the original action. If they say

Yes, the assistant continues with the step where the customer changed the topic. In the example, the assistant returns to the original action and repeats the question What is your CVV number?.

## Enabling and disabling changing the topic

By default, the **Change conversation topic** feature is enabled for all assistants and actions.

However, some processes are best completed without interruption, so you might want to disable this feature. You can disable for all actions, or just for an individual action.

To disable changing the topic for all actions:

1. From the **Actions** page of the assistant, click **Global settings** 
2. On the **Change conversation topic** tab, set the switch to **Off**.
3. Click **Save**, and then click **Close**.

To disable changing the topic for an individual action:

1. Edit an action, then click **Action settings** 
2. In the Action Settings window, toggle the **Change conversation topic** switch to **Off**.

## Confirmation to return to previous topic

By default, new assistants are set to ask a "yes or no" question to confirm that the customers want to return to the previous action. You can modify these settings.



**Note:** These settings aren't available when the assistant language is set to **Another language**, which uses the universal language model. For more information, see [Adding support for global audiences](#).

To change the confirmation settings:

1. From the **Actions** page of the assistant, click **Global settings** 
2. On the **Change conversation topic** tab, click the **Confirmation** section.

Settings that you can change are:

Setting	Description
Ask a confirmation question	Change the toggle to <b>Off</b> to disable the confirmation question for all actions.
Assistant says	By default, the assistant asks Do you want to return to your previous topic? and uses the system variable <code>Digressed from</code> to include the name of the previous action. Use the editor to modify the confirmation question.
Validation message	By default, the assistant says Please reply back with yes or no if the customer doesn't answer Yes or No. Use the editor to modify the validation message.
Show confirmation buttons	The assistant includes clickable Yes and No buttons with the confirmation question. Click the toggle to <b>Off</b> to disable the Yes and No buttons.

## Providing options when a question or request can't be answered

No matter how well your assistant is designed, sometimes customers can run into problems getting it to understand them or do what they want. Your assistant can automatically help customers recover from many such situations, and you can configure how it responds.

### How error conditions are handled

There are several kinds of error situations your assistant might need to recover from:

- Your assistant cannot understand your customer's request
- Your customer does not provide a valid response to a question
- Your customer asks to talk to a live agent

Your assistant can detect error conditions and give customers the chance to correct them. In addition, the built-in `Fallback` action provides a way to automatically connect customers to a live agent if they need more help.

### When the assistant can't understand your customer's request

When you build actions, you train your assistant on what your customers might ask for. The **Customer starts with** section of each action provides examples of customer input that trigger the action; the assistant uses natural language processing to recognize customer input that is similar to these examples. This happens at the beginning of the conversation, or after any action has completed and the assistant is ready for another action.

But you can't anticipate every possible request, so sometimes customers will send input that your assistant can't match to any action. This might happen because the input isn't phrased in a way that the assistant can understand; it can also happen if customers ask for things that your assistant isn't designed to handle.

Unrecognized input of this sort triggers the built-in `No action matches` action. To see how this action works, click **Set by assistant** in the list of actions, and then click `No action matches`.

No action matches built-in action

By default, this action has two steps, each step conditioned on the `No action matches count` session variable. This is a built-in variable that is automatically incremented with each consecutive unrecognized input. Therefore, the behavior of the `No action matches` action differs depending on how many times in a row the user has said something the assistant didn't understand:

- For the first three unrecognized messages, step 1 executes. This step simply outputs a message saying that the assistant did not recognize the user's input, and asking the user to try again. You can edit this step to change the message, or modify the step condition to change how many times the assistant responds with this message.

When the user sends input that successfully triggers an action, the `No action matches count` session variable is reset to 0.

- If the user tries more than three times but the assistant still doesn't understand, step 2 executes. Step 2 calls the `Fallback` action, which offers other options like connecting to a live agent. For more information about the `Fallback` action, see [Editing the fallback action](#).

**Tip:** You can edit the `No action matches` action just as you can any other action. This includes changing the existing steps and adding or deleting steps. Note that if you change the `No action matches` action, you might accidentally break your assistant's ability to recover from errors in the conversation. If this happens, you can recreate the default steps based on the information in this section.

## Adding examples of unsupported input

By default, the `No action matches` action is triggered only when the user's input does not match any defined action.

If there are certain user requests that you can anticipate, but that your assistant does not support, you can add these requests as example phrases in the **Customer starts with** section of the `No action matches` action. Adding example phrases helps to ensure that these requests are sent directly to the `No action matches` action rather than triggering a different action by mistake. You can also upload or download examples phrases in a comma-separated value (CSV) file, just like you can when building your own actions. For more information on uploading or downloading example phrases, see [Adding more examples](#).

## When your customer gives invalid answers

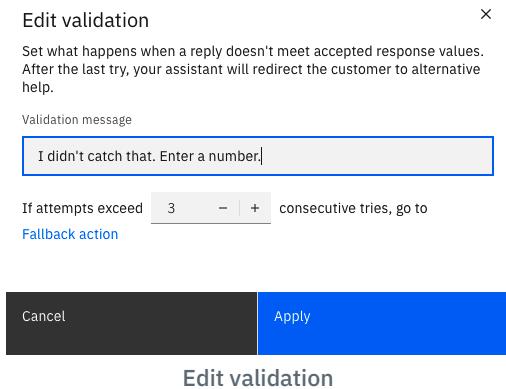
When a step in an action asks your customer to answer questions or provide additional information, the assistant expects a particular response type, such as a number, date, or text string. (For more information about customer response types, see [Collecting information from your customer](#).) The assistant checks the customer's response to make sure it fits the expected response type; this process is called *validation*.

For most customer response types, the assistant is able to understand valid responses provided in a variety of formats. For example, for a time value, `2:15 PM` and `a quarter past two in the afternoon` are both acceptable. But if the user provides a value that the assistant cannot interpret as matching the expected response type (for example, a response of `purple` when asked for a number), a validation error results.

When there is a validation error, the assistant asks the customer to try again. By default, the assistant allows three attempts to provide a valid response. After the third attempt, another invalid attempt triggers the `Fallback` action, which offers other options like connecting to a live agent. For more information about the `Fallback` action, see [Editing the fallback action](#).

## Customizing validation for a response

When you edit a step that expects a customer response, you can customize how validation errors are handled. Click **Edit validation** to see the validation options:



You can customize the following options:

- In the **Validation message** field, specify the text of the message the assistant sends when the customer's response doesn't match the expected response type. For example, the default validation error message for a numeric value is `I didn't catch that. Enter a number.` You might want to customize this message to be more specific (for example, `Enter the number of people in your group.`).
- Click `+` or `-`, or directly edit the number, to change how many consecutive tries the customer can make before the `Fallback` action is triggered.

## When your customer asks to speak to a live agent

At any point in the conversation, your customer might ask to speak to a live agent. The built-in `Fallback` action is preconfigured with example input that detects such requests; you can edit the **Customer starts with** section of the `Fallback` action to add more examples. You can also upload or download examples phrases in a comma-separated value (CSV) file, just like you can when building your own actions. For more information on uploading or downloading example phrases, see [Adding more examples](#).

## Editing the fallback action

The built-in `Fallback` action is automatically provided with each assistant and cannot be deleted. However, you can edit the `Fallback` action to modify the conversation your users have with the assistant when errors occur. For example, you might want to add steps or modify step conditions to provide more control over how specific error conditions are handled.

To edit the `Fallback` action, click **Set by assistant** in the list of actions, and then click `Fallback`.

**Fallback**

**Customer starts with:**  
Call agent

**Conversation steps**

1 **Fallback reason** is **Step validation failed**  
I'm afraid I don't understand. I can connect you to an agent.  
Connect to agent

2 **Fallback reason** is **Agent requested**  
Sorry I couldn't assist you. I will connect you to an agent right away.  
Connect to agent

3 **Fallback reason** is **No action matches**  
I am afraid I do not understand what you are asking, let me connect you to an agent.  
Connect to agent

4 **Fallback reason** is **Danger word detected**  
It seems this conversation would be best managed by a human agent. Let me connect you to one of...  
Connect to agent

5 **Fallback reason** is **Profanity detected**  
It seems this conversation would be best managed by a human agent. Let me connect you to one of...  
Connect to agent

**Action starts**

When your customer:

- Requests to connect to agent
- Fails step validation within an action
- Reaches the limit for **No action matches**

Use the assistant's default action or customize it.

**Additional training examples for connecting to an agent**

Tip: Add examples here to train your assistant on how your customer requests an agent.

Enter phrases your customer might use to start this action Total: 5

Enter a phrase

I would like to speak to someone

Can I connect to an agent?

I would like to speak to a human

Agent help

Call agent

### Fallback built-in action

Whenever the *Fallback* action is triggered, the assistant also sets a value for the *Fallback reason* session variable. This value indicates what kind of situation led to the *Fallback* action being triggered. By default, this variable can have one of five values:

- Step validation failed:** The customer repeatedly gave answers that were not valid for the expected customer response type.
- Agent requested:** The customer directly asked to be connected to a live agent.
- No action matches:** The customer repeatedly made requests or asked questions that the assistant did not understand.
- Danger word detected:** The customer uses words or phrases that match the *Connect to agent* step in the *Trigger word detected* action.
- Profanity detected:** The customer repeatedly used words or phrases that match the *Show warning* step in the *Trigger word detected* action.

For more information about the *Trigger word detected* action, see [Detecting trigger words](#).

The *Fallback* action defines five conditional steps, one for each possible value of the *Fallback reason* variable. Each step sends a message to the customer, based on the error condition, and then uses the *Connect to agent* feature to transfer the conversation to a live agent. (For more information about this feature, see [Connecting to a live agent](#).) You can modify these steps if you want to handle an error condition in a different way.

## Connecting to a live agent

Your assistant can do a lot, but there might be some situations when your customers need help from a human. If your assistant is integrated with one of the supported service desk systems, you can build in logic that will transfer the conversation to a live agent when necessary. This is referred to as an *escalation*.

To use this feature, your assistant must interact with customers using a **web chat** or **phone** integration. For more information, see [Deploying your assistant](#).

You must add an integration to a service desk system before setting up live agent transfers to your assistant. The supported service desks depend upon how customers connect to your assistant (web chat or phone). For more information about integrating with a service desk, see [Basic web chat configuration](#) and [Phone integration configuration](#).

Your agents can work with one of these supported service desk tools:

- [Genesys](#)
- [NICE CXone](#)
- [Salesforce](#)
- [Twilio Flex](#)
- [Zendesk](#)

There are two basic scenarios when your assistant might need to transfer a conversation to a live agent:

- *Planned escalation* refers to any anticipated situations in which you know you always want to hand off the conversation to a live agent.
- *Fallback escalation* is an unexpected situation in which the customer is unable to get help from the assistant.

When the assistant initiates a transfer, the agent receives a notification within the agent dashboard, and has access to the history of the customer's chat with the assistant.

## Planned escalations

Examples of planned escalations might include the following:

- The customer asks for a service that cannot be completed without the assistance of a live agent
- The customer needs help with a sensitive subject that requires a human touch, such as asking about bereavement benefits or resolving a complaint

To set up a planned escalation, you build an action that can recognize a specific situation that requires a live agent. An example would be an action that is triggered by customer input `I want to pay my bill` (you might want to let live agents handle payments).

Within any action, you can create a step that initiates a transfer to a live agent:

1. Add a step or edit an existing step to transfer the conversation to a live agent.

Transferring the conversation to a live agent ends the action. If there are situations where you want the conversation to continue within the assistant rather than being transferred, use step conditions as needed.

2. In the **And then** field at the end of the step, select **Connect to agent**.

3. In the **Settings** window, you can customize messages the assistant displays as part of the transfer:

Settings

## Connect to agent

This will transfer customers based on the integrations you've set up with your assistant. [Learn more](#)

Response if agents are online

Let's send you to an available agent.

Response if agents are offline

There are no agents available at this time. When one becomes available, we'll connect you.

Message to agent (Optional)

Example: Unable to resolve. See history.

Route to specific queue (Optional)

Nice CXOne



[Set attribute](#) [+](#)

Cancel

Appl

- **Response if agents are online:** The message the assistant sends to the customer when the conversation is being transferred to an agent. The default message is Let's send you to an available agent.
- **Response if agents are offline:** The message the assistant sends to the customer when no agents are currently available to take over the conversation. The default message is There are no agents available at this time. When one becomes available, we'll connect you.
- **Message to agent:** An optional message the assistant sends to the live agent when transferring the conversation.
- **Route to a specific queue:** An optional selection to route customers to a specific integration, which can be helpful if you have more than one set up.

4. Click **Apply**.



**Note:** If you want to edit the transfer settings later, click **Edit settings** in the **And then** field.

## Fallback escalations

Examples of fallback escalations include:

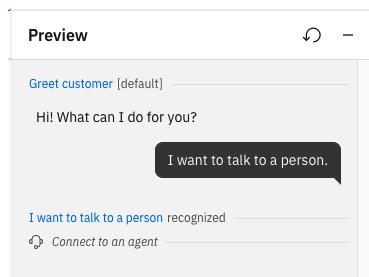
- The customer repeatedly asks a question or makes a request that the assistant cannot match to any defined action.
- The customer repeatedly gives an invalid answer to a question.
- The customer explicitly asks to speak to a human.

Fallback escalations use the *Fallback* action, which is a built-in system action that is automatically triggered in any of these fallback scenarios. By default, the *Fallback* action handles these error conditions by initiating a transfer to a live agent.

For more information about this automatic error handling and the *Fallback* action, see [Handling errors in the conversation](#).

## Testing the transfer in the Preview pane

After you have configured an action to connect to a live agent, you can preview it by clicking **Preview**. Note that in the action Preview pane, no actual transfer takes place, but the *Connect to an agent* message confirms that it was correctly triggered.



If you want to test the actual transfer using a working service desk integration, you can do so using the assistant Preview page. For more information, see [Previewing and sharing your assistant](#).

## Adding conditions to an action

An action condition is a boolean test, based on some runtime value; the action executes only if the test evaluates as true. This test can be applied to any variable. By defining action conditions, you can do things such as control user access to actions or create date-specific actions.

This is a beta feature that is available for evaluation and testing purposes.

For more information about variables, see [Using variables to manage conversation information](#).

A basic action condition is expressed in the following form:

If {variable} {operator} {value}

where:

- {variable} is the name of a variable or an expression.
- {operator} is the type of test to apply to the variable value (for example, `is` or `is not`).
- {value} is the value to compare to the variable.

For example, an action condition might read:

If User category? is employee

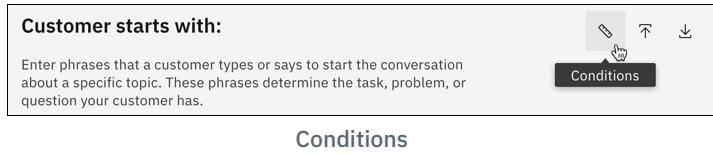
In this example, `User category` can be a list of employees at your organization. This condition evaluates as true if the user is an active employee. If false, you can control access so that former employees can't use the action.

Conditions can be grouped together to construct complex tests.

To add an action condition:

1. In an action, click **Customer starts with**.

2. Click **Conditions**.



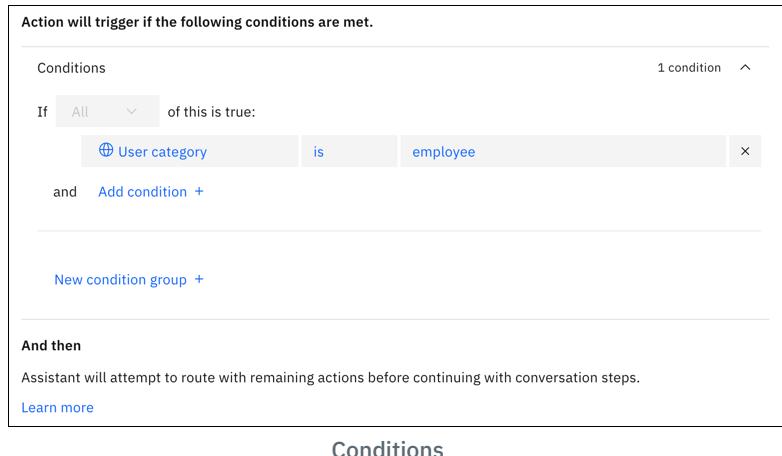
3. Choose the variable for the condition. You can select any of the following:

- An action variable storing the customer response from a previous step in the action
- A session variable containing a value stored by any action
- A built-in variable set by the assistant or by an integration

**Note:** You can also define a complex condition by writing an expression defining some other value. For more information about expressions, see [Writing expressions](#).

4. Select the operator representing the test you want to perform on the variable (for example, `is` or `is not`). The available operators for a particular value depend upon its data type. For more information, see [Operators](#).

5. Select the value you want to evaluate the condition against. The values available depend upon the type of value you are testing. For example, a variable containing an options response can be tested against any of the defined options, and a date value can be tested against any date.



6. To add more than one condition to an action, click **Add condition**.

7. To add another group of conditions, click **New condition group**.

You can use groups to build complex action conditions. Each group is evaluated true or false as a whole, and then these results are evaluated together. For example, you might build an action that executes only if all conditions in group 1 are true or any condition in group 2 is true. (Groups function like parentheses in the boolean conditions of many programming languages.)

After you add a group, you can define one or more conditions in the new group. Between groups, choose **and** or **or** to indicate

whether the conditions in both conditional groups or only one of them must be met for the step to be included in the conversational flow.

## Operators

An operator specifies the kind of test you are performing on a value in a condition. The specific operators available in a condition depend on the customer response type of the value, as shown in the following table.

Response type	Operators
• Options	<ul style="list-style-type: none"><li>• is</li><li>• is not</li><li>• is any of</li><li>• is none of</li></ul>
• Regex	<ul style="list-style-type: none"><li>• is</li><li>• is not</li></ul>
• Number	<ul style="list-style-type: none"><li>• is defined</li></ul>
• Currency	<ul style="list-style-type: none"><li>• is not defined</li></ul>
• Percent	<ul style="list-style-type: none"><li>• is equal to (==)</li><li>• is not equal to (≠)</li><li>• is less than (&lt;)</li><li>• is less than or equal to (&lt;=)</li><li>• is greater than (&gt;)</li><li>• is greater than or equal to (&gt;=)</li></ul>
• Date	<ul style="list-style-type: none"><li>• is defined</li><li>• is not defined</li><li>• is on (also allows specific day of the week)</li><li>• is not on</li><li>• is before</li><li>• is after</li><li>• is on or before</li><li>• is on or after</li></ul>
• Time	<ul style="list-style-type: none"><li>• is defined</li><li>• is not defined</li><li>• is at</li><li>• is not at</li><li>• is before</li><li>• is after</li><li>• is at or before</li><li>• is at or after</li></ul>

- Free text
- is
- is not
- contains
- does not contain
- matches
- does not match

## Operators

## Detecting trigger words

Use the *Trigger word detected* action to add words or phrases to two separate groups. The first group connects customers with an agent. The second group shows customers a customizable warning message.

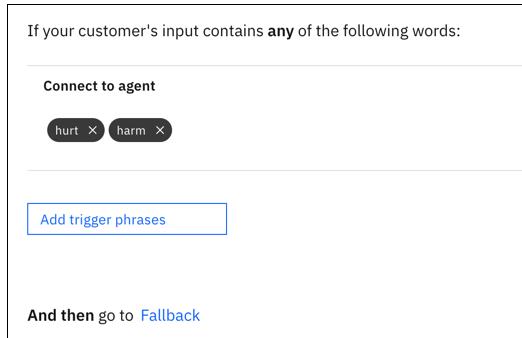
By default, this action has two steps—the *Connect to agent* step and the *Show warning* step. To see how this action works, click **Set by assistant** in the list of actions, and then click **Trigger word detected**.

This is a beta feature that is available for evaluation and testing purposes.

### Connect to agent

The first step of the *Trigger word detected* action is the *Connect to agent* step. The *Connect to agent* step goes to the *Fallback* action if any trigger words are detected in the customer's input. Use this step to capture any key phrases where it's important to connect a customer with a live agent rather than activate any further actions.

For example, you might add `hurt` and `harm` as trigger words for the *Connect to agent* step:



Adding trigger words to the *Connect to agent* step

In this example, a customer enters a word or phrase including `hurt` or `harm`, which triggers the *Fallback* action. Step 4 has:

- Danger word detected as the fallback reason.
- The default message: It seems this conversation would be best managed by a human agent. Let me connect you to one of our agents. You can customize this message.
- **And then** set to **Connect to agent (action ends)**.

For more information about the *Fallback* action, see [Editing the fallback action](#).

### Show warning

The second and final step of the *Trigger word detected* action is the *Show warning* step. The *Show warning* step shows a customizable warning message to your customer if any trigger words are detected in the customer's input. Use this step to discourage customers from interacting with your assistant in unacceptable ways, such as using profanity.

For example, you might add `darn`, `dang`, and `heck` as trigger words for the *Show warning* step:

If your customer's input contains **any** of the following phrases:

**Show warning**

darn × dang × heck ×

[Add trigger phrases](#)

**Assistant says**

Please use appropriate language when interacting with the assistant.

**And then** return to action

If attempts exceed  - + total tries, go to [Fallback](#)

Adding trigger words to the *Show warning* step

In this example, a customer enters `darn`, `dang`, or `heck`, the assistant responds with `Please use appropriate language when interacting with the assistant`. You can customize this message.

If the customer triggers the *Show warning* step again, the *Fallback* action is triggered. The default setting is if attempts exceed 2 total tries. You can customize this setting.

In the *Fallback* action, step 5 has:

- Profanity detected as the fallback reason.
- The default message: It seems this conversation would be best managed by a human agent. Let me connect you to one of our agents. You can customize this message.
- **And then** set to **Connect to agent (action ends)**.

For more information about the *Fallback* action, see [Editing the fallback action](#).

## Saving your actions

When working on actions, your assistant automatically saves changes when you do one of the following:

- Click on a new step
- Open Preview
- Reset Preview

You can also save changes by clicking the **Save** icon, for example, in these places when working on actions:

- Within each action
- Actions settings
- Renaming an action
- Editing a variable

When the system automatically saves or you click the **Save** icon, the following items are saved to all:

- Actions you have created
- Edits you made to a system action
- Variables you created
- Action settings

If you want to disable auto-save, you can toggle it off:

1. Open the **Actions** page.
2. Click the **Global settings** icon .
3. On the **Auto-save** tab, set the switch to **Off**.
4. Click **Save**, and then click **Close**.

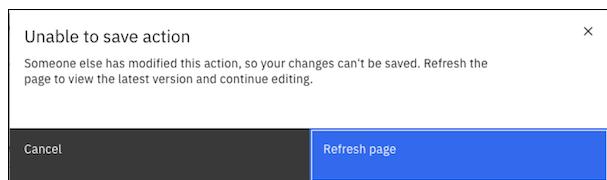


**Note:** The auto-save setting is a user preference that applies to all assistants in your instance. Changing the setting applies only to you and doesn't affect other users.

## Avoiding conflicts

To avoid conflicts when saving, we recommend dividing work so that one person works on one action at a time.

Also, Watson Assistant prevents two users from working in an action with unsaved edits. If one user saves changes before a second user, this `Unable to save action` message may appear:



## Reviewing and debugging your actions

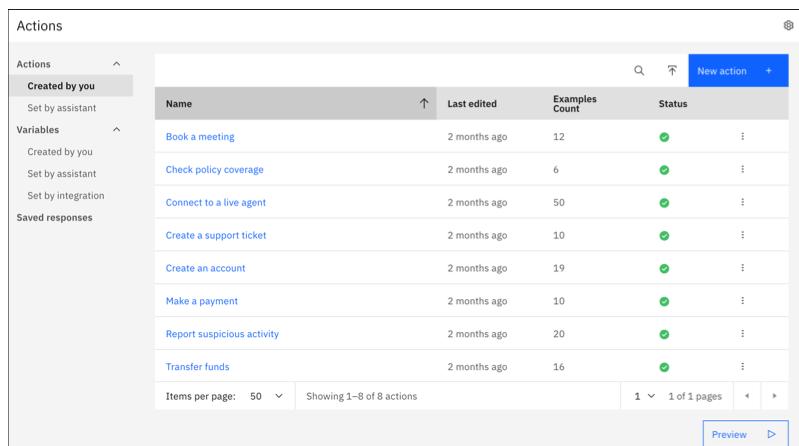
Learn how to test the conversation you built into an action, to experience what your users see with your assistant. If there are any issues, learn how to debug the user's experience.

### Using Preview to test your action

As you make changes, test the action at any time to see whether the resulting interaction works as intended. **Preview**, which is a button on every action page, shows you what customers see when they use web chat to interact with the action.



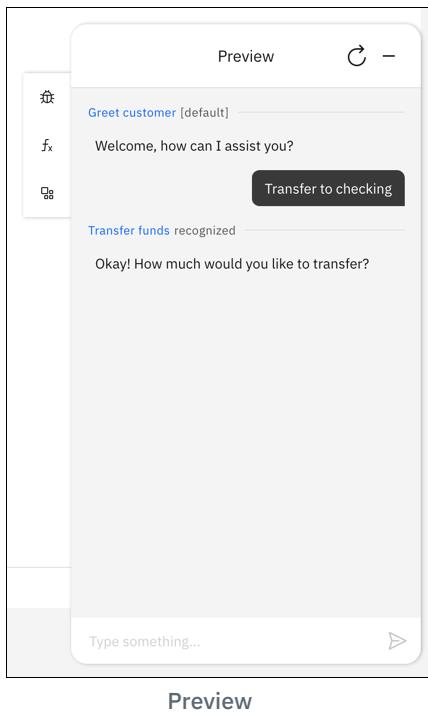
**Note:** Before you test your action, make sure you save any new changes, and wait until the system finishes training. If the system is still training, a message is displayed that says so.



Preview button

1. Click **Preview**. The Greet customer action starts.
2. In the chat window, type some text and then press Enter.
3. Check the response to see if your assistant correctly interpreted the input, started the intended action, and performed the appropriate step.

The Preview pane names the action that was recognized in the input.



### Preview

If the assistant doesn't understand a phrase, you see the built-in action `No action matches`.

4. Continue to converse with your assistant to see how the conversation flows.
5. To remove prior test utterances from the chat pane and start over, click the **Reset** icon. Not only are the test utterances and responses removed, but this action also clears the values of any variables that were set as a result of your previous interactions.

**Note:** Queries that you submit through the Preview pane generate `/message` API calls, but they are not logged and do not incur charges.

## Saving changes before testing

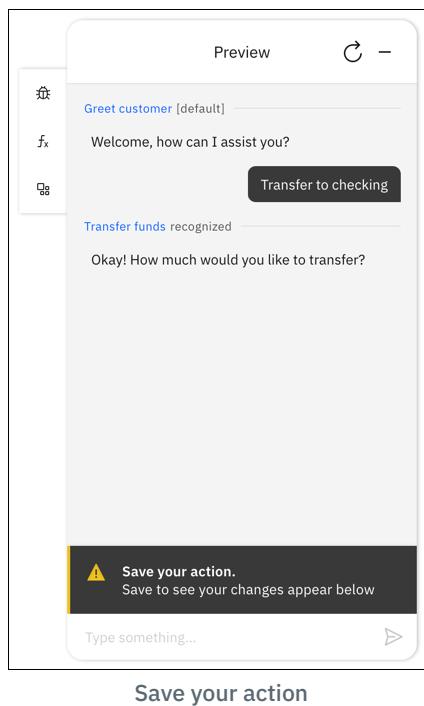
**Preview** represents updates from the last time that the assistant was saved.

Changes are saved when you:

- Select the save icon
- Click a new step
- Open Preview
- Reset Preview

To learn more about saving changes, see [Saving your work](#).

If you make several edits without saving, the preview pane shows a message that you need to save before you test your changes.



Save your action

## Using debug mode in Preview

Preview has a debug mode that you can turn on to see information that helps you understand why the assistant responds or doesn't respond to a particular input.

Debug mode has four tools to analyze your action:

- [Start and end of an action](#)
- [Confidence scores](#)
- [Step locator](#)
- [Follow along](#)

### Start and end of an action

The assistant marks the spots in the conversation when a customer enters an input that fits within an action. The assistant also marks when an action completes, and how it completes.

Completion options include ending:

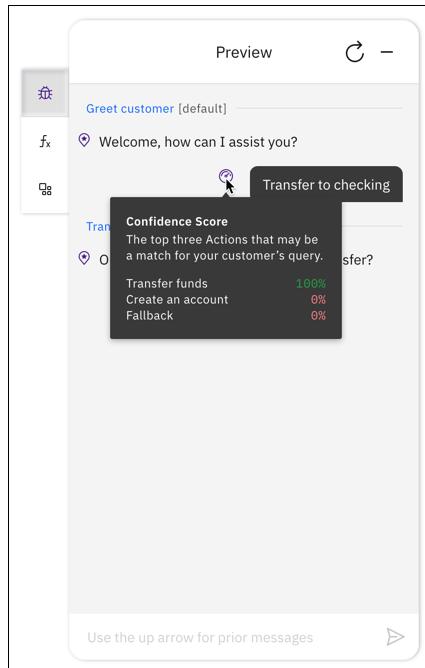
- With an end step
- Without an end step
- With a human agent escalation
- With a search to a knowledge base

### Action confidence score

Every input that you enter that can start a new topic shows a confidence score icon. Hover over this icon to see a list of actions with different confidence scores.

These scores represent the assistant's confidence that the sentence or phrase that you entered can be solved by the steps that

are built into a specific action.



Debug mode

The top score in green represents the action with the highest confidence and the one the assistant used.

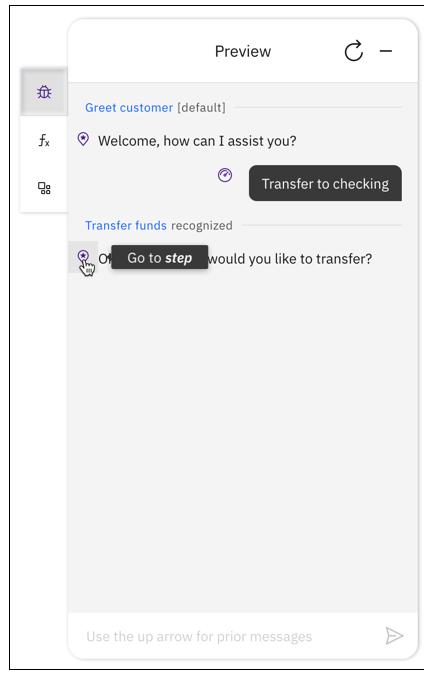
The remaining two are actions that were considered because of their confidence score, but weren't used because the confidence scores were lower.

If no action scores higher than 20% confidence, you see the built-in action `No action matches`.

## Step locator

Sometimes you might find an error in the middle of a test conversation, and need to find which step and action is involved. A locator icon next to each assistant response lets you find the associated steps in the editor.

Click the icon, and the editor shows the corresponding step in the background.



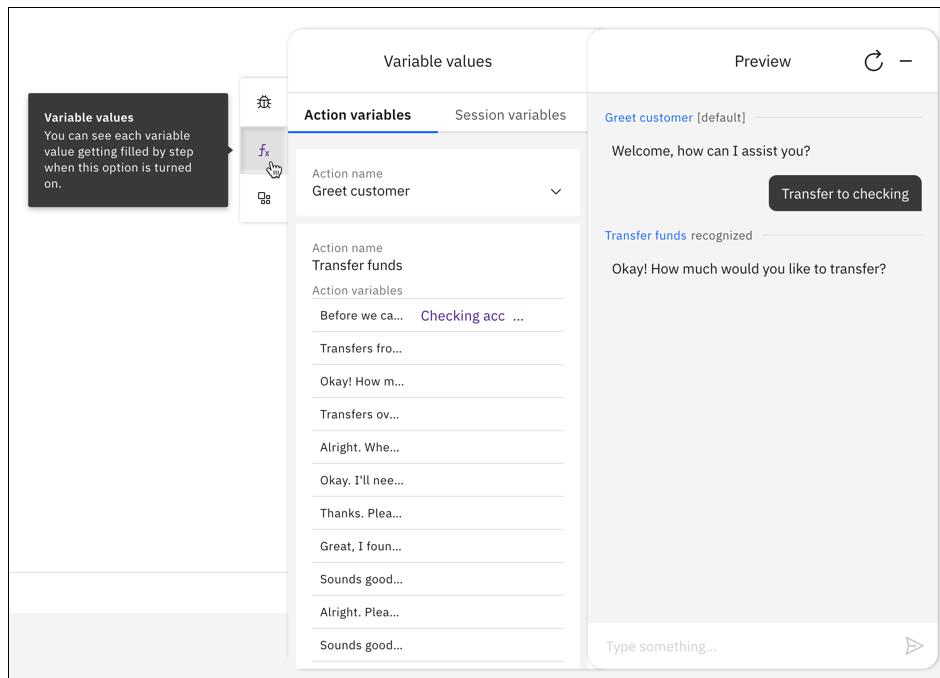
Step locator

## Follow along

**Follow along** connects what you are seeing in Preview with what you built in the action. As you interact with your assistant, the debug mode automatically opens each step in the background. That means you can fix an error as soon as you see it, because the editor is already open to the corresponding step.

## Variable values in Preview

As you test your conversation in Preview, you can check that each variable is set correctly. Click **Variable values** to see the values stored in each variable during the conversation. The **Variable values** pane has two tabs, one for action variables and one for session variables. If you are using dialog, you can see session variables for both actions and dialog on the **Session variables** tab.

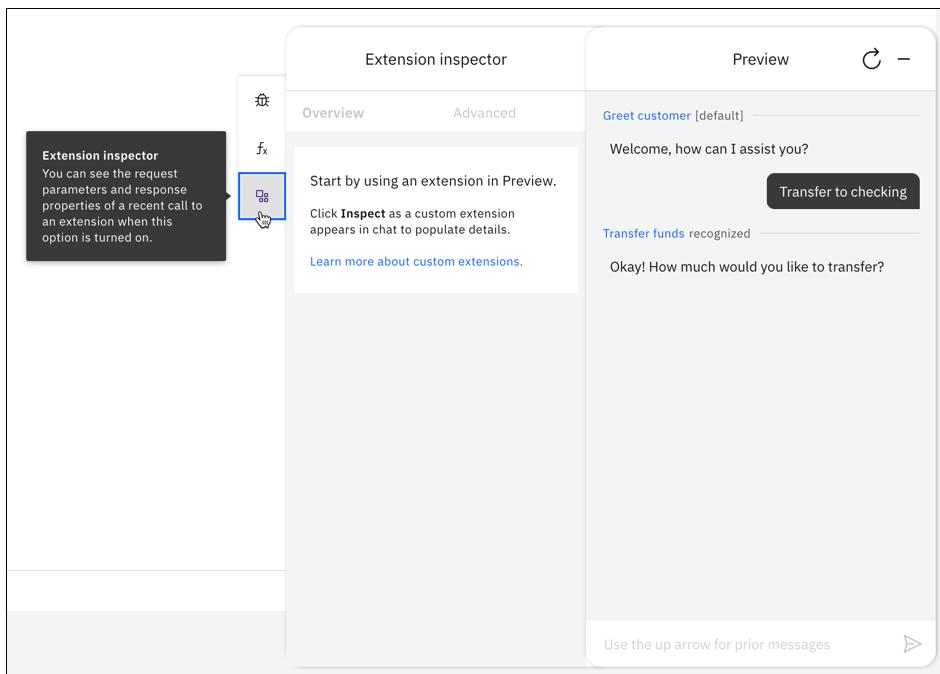


Variable values

To learn more about variables, see [Managing information during the conversation](#).

## Extension inspector in Preview

If you are using a custom extension in your action, you can use the **Extension inspector** in Preview to debug. For more information, see [Debugging custom extensions](#)



Extension inspector

## Global settings for actions

Use **Global settings** to configure features across all actions.

On the **Actions** page, click **Global settings** .

This page provides options, configurations, and tasks for the following:

- [Ask clarifying question](#)
- [Change conversation topic](#)
- [Autocorrection](#)
- [Display formats](#)
- [Algorithm version](#)
- [Auto-save](#)
- [Upload/Download](#)

## Ask clarifying question

When your assistant finds that more than one action might fulfill a customer's request, it can automatically ask for clarification. Instead of guessing which action to take, your assistant shows a list of the possible actions to the customer, and asks the customer to pick the right one. For more information, see [Asking clarifying questions](#).

On the **Ask clarifying question** tab, you can make the following changes:

- In the **Assistant says** field, edit the text that is displayed before the list of clarification choices.

The default text is *Did you mean:*. You can change it to something else, such as *What do you want to do?* or *Pick what to do next.*

- In the **Label for a fallback choice** field, edit the label that is displayed for the choice that customers can click when none of the other choices are quite right. When a user picks this choice, the *No action matches* system action is taken next.

The label *None of the above* is used if you don't change it.

This fallback choice gives customers a way to get out of the clarification process if it's not helping them. If you don't want to give customers a fallback choice, remove the text from the field.

If necessary, you can disable clarifying questions for all actions. To disable clarification for all actions:

1. On the **Ask clarifying question** tab, set the switch to **Off**.
2. Click **Save**, and then click **Close**.

## Change conversation topic

The **Change conversation topic** feature enables your assistant to handle digressions, dynamically responding to the user by changing the conversation topic as needed. For more information, see [Allowing your customers to change the topic of the conversation](#).

If necessary, you can disable changing the topic for all actions:

1. On the **Change conversation topic** tab, set the switch to **Off**.
2. Click **Save**, and then click **Close**.

## Autocorrection

*Autocorrection* fixes misspellings that users make in their requests. The corrected words are used to match to an action.

Autocorrection is enabled automatically for all English-language assistants. It is also available in French-language assistants, but is disabled by default. The autocorrection setting isn't available for any other languages.

For more information, see [Autocorrecting user input](#).

## Display formats

**Display formats** lets you specify the display formats for variables that use date, time, numbers, currency, or percentages. You can also choose a default locale to use if one isn't provided by the client application. This lets you make sure that the format of a variable that's displayed in the web chat is what you want for your assistant. For example, you can choose to have the output of a time variable appear in HH:MM format instead of HH:MM:SS.

Variables are formatted using a system default unless you specify otherwise.

Display format setting	Description	English - United States (en-US) examples
Locale	Choose a default locale for the assistant if one can't be determined. The locale you choose uses formats specific to that country and language. If you choose a locale, the date, time, number, currency, and percentage format fields change to show choices specific to that locale. The system default is English - United States (en-US).	
Date	For calendar dates, choose short, medium, long, full, YY/MM/DD, or YYYY/MM/DD	<ul style="list-style-type: none"><li>• Short: 1/31/23</li><li>• Medium: Jan 31, 2023</li><li>• Long: January 31, 2023</li><li>• Full: Tuesday, January 31, 2023</li></ul>
Time	For times, choose short or medium	<ul style="list-style-type: none"><li>• Short: 1:53 PM</li><li>• Medium: 1:53:30 PM</li></ul>
Number fraction digits	Use the system default (up to 14 digits) or two digits	10.12
Number delimiter	Use the system default, none, comma, or period	<ul style="list-style-type: none"><li>• None: 2000.12</li><li>• Comma: 2,000.12</li><li>• Period: 2.000,12</li></ul>
Currency fraction digits	Use the system default (up to 14 digits) or two digits	10.99
Currency symbol	Use the system default or choose a global symbol	\$10.99
Percentage fraction digits	Use the system default (up to 14 digits) or two digits	10.75%

## Algorithm version

**Algorithm version** allows you to choose which Watson Assistant algorithm to apply to your future trainings. For more information, see [Algorithm version](#).

## Auto-save

When working on actions, your assistant automatically saves changes when you do one of the following:

- Click on a new step
- Open Preview
- Reset Preview

For more information, see [Saving your actions](#)

If you want to disable auto-save, you can toggle it off:

1. On the **Auto-save** tab, set the switch to **Off**.
2. Click **Save**, and then click **Close**.



**Note:** The auto-save setting is a user preference that applies to all assistants in your instance. Changing the setting applies only to you and doesn't affect other users.

## Upload/Download

You can upload or download actions.

### Downloading

To back up actions, download a JSON file and store it. On the **Upload/Download** tab, click the **Download** button.

### Uploading

To reinstate a backup copy of actions that you exported from another service instance or environment, import the JSON file of the actions you exported.



**Important:** If the Watson Assistant service changes between the time you export the actions and import it, due to functional updates that are regularly applied to instances in cloud-hosted continuous delivery environments, your imported actions might function differently than before.

On the **Upload/Download** tab, drag and drop a JSON file onto the tab or click to select a file from your local system, then click **Upload**.



**Important:** The imported JSON file must use UTF-8 encoding, without byte order mark (BOM) encoding. The JSON file cannot contain tabs, newlines, or carriage returns.

## Autocorrecting user input

*Autocorrection* fixes misspellings that users make in their requests. The corrected words are used to match to an action or an intent.

Autocorrection corrects user input in the following way:

- Original input: `letme applt for a memberdhip`

- Corrected input: let me apply for a membership

When your assistant evaluates whether to correct the spelling of a word, it does not rely on a simple dictionary lookup process. Instead, it uses a combination of natural language processing and probabilistic models to assess whether a term is, in fact, misspelled and should be corrected.

Autocorrection is enabled automatically for all English-language assistants. It is also available in French-language assistants, but is disabled by default. Autocorrection isn't available for any other languages.

## Disabling autocorrection

If necessary, you can disable autocorrection for your assistant.

 **Note:** If you find that a domain-specific term is being corrected that shouldn't be, you can prevent the correction from happening by adding the term or phrase to your training data. For more details, see [Autocorrection rules](#).

If you are using actions in your assistant, follow these steps to disable autocorrection:

1. On the **Actions** page, click **Global settings** .
2. Click the **Autocorrection** tab.
3. Set the switch to **Off**, then click **Save**.

If you are using dialog in your assistant, follow these steps to disable autocorrection:

1. In the **Options** section, click **Autocorrection**.
2. Set the switch to **Off**.

## Testing autocorrection in dialog

If you are using dialog, you can test autocorrection using **Try it out**.

1. In **Try it out**, enter a request that includes some misspelled words.

If words in your input are misspelled, they are corrected automatically, and an  icon is displayed. The corrected utterance is underlined.

2. Hover over the underlined utterance to see the original wording.

If there are misspelled terms that you expected your assistant to correct, but it did not, then review the rules that your assistant uses to decide whether to correct a word to see if the word falls into the category of words that your assistant intentionally does not change.

## Autocorrection rules

To avoid overcorrection, your assistant does not correct the spelling of the following types of input:

- Capitalized words
- Emojis
- Locations, such as states and street addresses
- Numbers and units of measurement or time
- Proper nouns, such as common first names or company names
- Text within quotation marks

- Words containing special characters, such as hyphens (-), asterisks (\*), ampersands (&), or at signs (@), including those used in email addresses or URLs.
- Words that *belong*, meaning words that have implied significance because they occur in your action steps or dialog entity values, entity synonyms, or intent user examples.

## How is spelling autocorrection related to fuzzy matching?

In dialog, *fuzzy matching* helps your assistant recognize dictionary-based entity mentions in user input. It uses a dictionary lookup approach to match a word from the user input to an existing entity value or synonym in the skill's training data. For example, if the user enters `boook`, and your training data contains a `@reading_material` entity with a `book` value, then fuzzy matching recognizes that the two terms (`boook` and `book`) mean the same thing.

In dialog, when you enable both autocorrection and fuzzy matching, the fuzzy matching function runs before autocorrection is triggered. If it finds a term that it can match to an existing dictionary entity value or synonym, it adds the term to the list of words that *belong* to the skill, and does not correct it.

For example, if a user enters a sentence like `I wnt to buy a boook`, fuzzy matching recognizes that the term `boook` means the same thing as your entity value `book`, and adds it to the protected words list. Your assistant corrects the input to be, `I want to buy a boook`. Notice that it corrects `wnt` but does *not* correct the spelling of `boook`. If you see this type of result when you are testing your dialog, you might think your assistant is misbehaving. However, your assistant is not. Thanks to fuzzy matching, it correctly identifies `boook` as a `@reading_material` entity mention. And thanks to autocorrection revising the term to `want`, your assistant is able to map the input to your `#buy_something` intent. Each feature does its part to help your assistant understand the meaning of the user input.

## Algorithm version and training

---

The **Algorithm version** setting allows you to choose which Watson Assistant algorithm is used for training. Your assistant is trained when you update the content or settings, or through [automatic retraining](#).

There are three choices:

- **Beta:** Use this to preview and test what is coming. The capability in the beta version at any given time is likely to become a supported version later on. It's not recommended to use the beta version in a production deployment.
- **Latest:** The current supported version that's recommended for your live production assistant.
- **Previous:** The last supported before the latest version. Support for this version ends when the next version is released.

To choose an algorithm version for actions:

1. On the **Actions** page, click **Global settings** .
2. Click the **Algorithm Version** tab.
3. Choose a version, then click **Save**.

To choose an algorithm version for dialog:

1. On the **Dialog** page, open the **Options** section.
2. Click **Algorithm Version**.
3. Choose a version.

The latest and previous versions have date labels such as **Latest (01-Jun-2022)** or **Previous (01-Jan-2022)**. See the [Watson Assistant release notes](#) for details about each algorithm version release.

Algorithm version choices are currently available for Arabic, Chinese (Simplified), Chinese (Traditional), Czech, Dutch, English,

French, German, Japanese, Korean, Italian, Portuguese, and Spanish. The universal language model uses a default algorithm.

## Automatic retraining



IBM Cloud only

Watson Assistant was released as a service in July 2016. Since then, users have been creating and updating skills to meet their virtual assistant needs. Behind the scenes, Watson Assistant creates machine learning (ML) models to perform a variety of tasks on the user's behalf.

The primary ML models deal with action recognition, intent classification, and entity detection. For example, the model might detect what a user intends when saying `I want to open a checking account`, and what type of account the user is talking about.

These ML models rely on a sophisticated infrastructure. There are many intricate components that are responsible for analyzing what the user has said, breaking down the user's input, and processing it so the ML model can more easily predict what the user is asking.

Since Watson Assistant was first released, the product team has been making continuous updates to the algorithms that generate these sophisticated ML models. Older models have continued to function while running in the context of newer algorithms. Historically, the behavior of these existing ML model did not change unless the skill was updated, at which point the skill was retrained and a new model generated to replace the older one. This meant that many older models never benefited from improvements in our ML algorithms.

Watson Assistant uses continuous retraining. The Watson Assistant service continually monitors all ML models, and automatically retrains those models that have not been retrained in the previous 6 months. Watson Assistant retrains using the selected algorithm version. If the version you had selected is no longer supported, Watson Assistant retrains using the version labeled as **Previous**. This means that your assistant will automatically have the supported technologies applied. Assistants that have been modified during the previous 6 months will not be affected.



**Note:** In most cases, this retraining will be seamless from an end-user point of view. The same inputs will result in the same actions, intents, and entities being detected. In some cases, the retraining might cause changes in accuracy.

## Instructions for Watson Assistant for IBM Cloud Pak for Data



When upgrading your instance of Watson Assistant for IBM Cloud Pak for Data, as long as your existing models have been trained using an algorithm version that is still supported, your models will not be retrained during or after the upgrade.

New algorithm versions will be included in major releases (for example, 4.0.0 or 5.0.0) or minor releases (for example, 4.5.0 or 4.6.0). A monthly release might include a new algorithm version if there is more than 6 months between major or minor releases. Each algorithm version supports 2 major/minor releases or a maximum time of 12 months, whichever is first. For more information, see the [IBM Cloud Pak for Data Software Support Lifecycle Addendum](#)

Each new release includes full support for the version listed as **Latest** in the most recent prior release. This version is then labeled as **Previous** after you upgrade. In addition, each new release will support running models that were trained on the version prior to that so that upgrading won't impact your runtime. For example, if you upgrade from IBM Cloud Pak for Data 4.6 to 4.7, and were using **Latest (01-Jun-2022)** that version becomes listed as **Previous (01 June 2022)** and remains your selected version.

## Automatic retraining after upgrading

After your Watson Assistant for IBM Cloud Pak for Data upgrade is complete, Watson Assistant performs automatic retraining for any assistant models that were trained using a version that is no longer supported. In this case, Watson Assistant automatically retrains your assistant to the **Latest** version. This automatic retraining is required to assure your ability to run your trained models in your next upgrade.

## Best practices

It's recommended to use the **Latest** version in your production deployment of Watson Assistant for IBM Cloud Pak for Data. This is the default for newly-created assistants. During an upgrade, your settings don't automatically switch existing assistants to use the latest version. If prior to your upgrade you had selected **Latest**, your settings continue to use that version, now labeled as **Previous**. After you upgrade, it's recommended you choose **Latest** and run basic regression tests.

IBM performs robust testing on a variety of data sets to minimize impacts on existing assistants. But given the nature of machine learning models and the nuance and subtlety of natural languages processing, you may find some discrepancies from version to version. If you find a major issue through your tests, you have the ability to switch your settings and use **Previous** to return to the prior behavior. In this event, we recommend you contact IBM and provide details of your test so that that IBM can support you in the steps to resolve the problem.

It's also recommended that you try the **Beta** version in one of your test systems after you upgrade. This gives you early visibility to changes that are likely to be delivered in a future version, and will reduce the probability of negative impacts to your production systems. IBM values both positive and negative feedback from customers who use Beta. You will have the opportunity to shape how the algorithms function before the version is promoted to **Latest** in a future version. If you choose **Beta**, your assistant always trains on the most current beta version.

## Managing actions

---

Use different Watson Assistant capabilities to manage your actions workflow within a single assistant and between multiple assistants.

### Duplicating an action

You can duplicate an action to reuse information in a new action. When you duplicate an action, the new action includes everything except example phrases.

To duplicate an action, click the overflow menu on the action you want and select **Duplicate**.



### Copying an action to another assistant

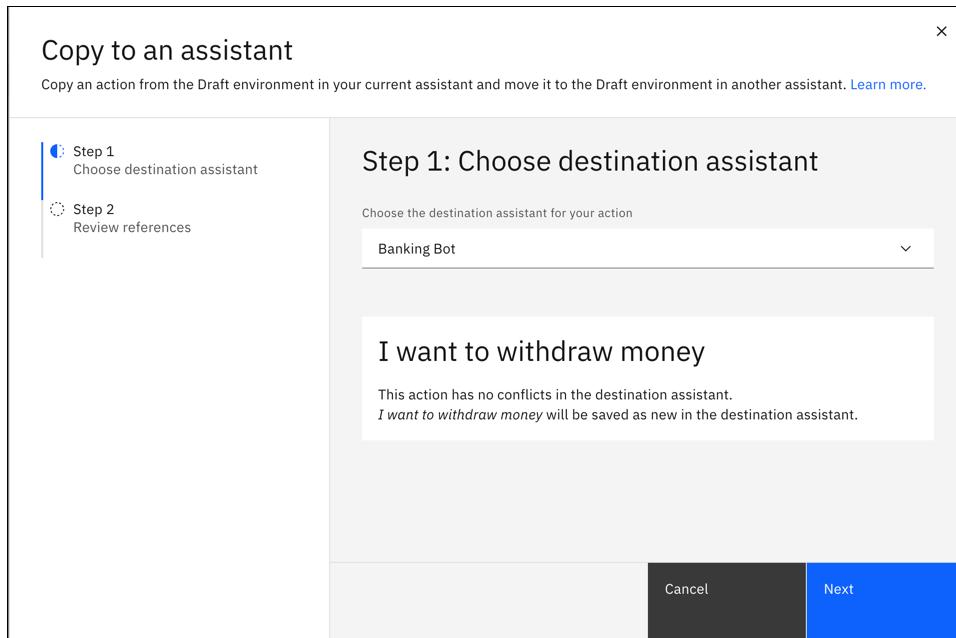
You can copy an action from one assistant to another. When you copy an action, references to other actions, variables, and saved responses are also copied.

To copy an action to another assistant:

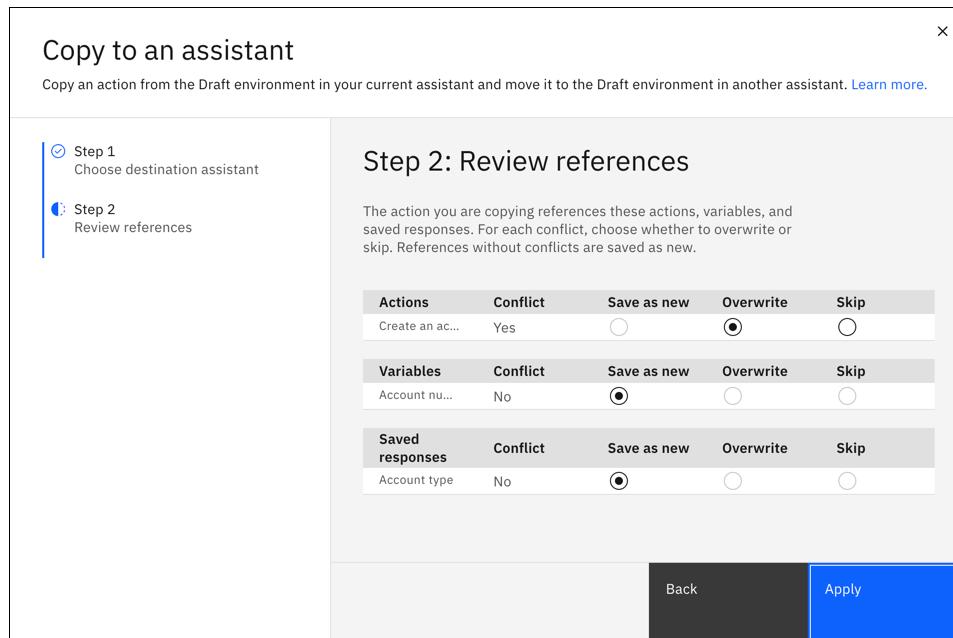
1. Click the overflow menu on the action you want and select **Copy to an assistant**.

Name	Last edited	Examples Count	Status
Create a support ticket	3 months ago	10	✓
Create an account	2 months ago	19	✓
I want to withdraw money	an hour ago	1	✓
Make a payment	2 months ago	10	
Report suspicious activity	2 months ago	20	
Transfer funds	3 months ago	16	

2. Choose the destination assistant for the copy. If the action doesn't already exist in the destination assistant, it's copied as new. If the action already exists and there is a conflict, the copy overwrites the existing one.



3. If there are references to other actions, variables, or saved responses, you can review the list. References without a conflict are saved as new. If there are conflicts, you can choose to overwrite the existing reference, or skip copying it. If there are no references, this step doesn't appear.



4. Click **Apply** to finish copying the action to the other assistant.

## Uploading or downloading all actions

You can upload or download all your actions as a JSON file.

### Downloading

To back up all your actions, download a JSON file and store it.

1. On the **Actions** page, click **Global settings** .
2. On the **Upload/Download** tab, click the **Download** button.

### Uploading

To reinstate a backup copy of actions that you exported from another service instance or environment, import the JSON file of the actions you exported.

 **Important:** If the Watson Assistant service changes between the time you export the actions and import it, due to functional updates that are regularly applied to instances in cloud-hosted continuous delivery environments, your imported actions might function differently than before.

1. On the **Actions** page, click **Global settings** .
2. On the **Upload/Download** tab, drag and drop a JSON file onto the tab or click to select a file from your local system, then click **Upload**.

 **Important:** The imported JSON file must use UTF-8 encoding, without byte order mark (BOM) encoding. The JSON file cannot contain tabs, newlines, or carriage returns.

## Uploading intents as actions

If you created intents in the classic Watson Assistant experience, you can migrate your intents to actions in the new Watson Assistant experience. When you upload intents, each intent is created as a new action. All phrases corresponding to an intent are created as example phrases for the new action. This can provide a helpful starting point when you are ready to start building actions in the new experience.

1. Download the intents that you want to migrate to actions from the classic Watson Assistant experience. For more information, see [Downloading intents](#). The format for each line in the file is as follows:

```
$ <phrase>,<intent>
```

where `<phrase>` is the text of a user example phrase, and `<intent>` is the name of the intent. For example:

```
$ Tell me the current weather conditions.,weather_conditions
  Is it raining?,weather_conditions
  What's the temperature?,weather_conditions
  Where is your nearest location?,find_location
  Do you have a store in Raleigh?,find_location
```

2. From the main actions page, click the **Upload** icon .

3. Select the intents file that you downloaded.

The file is validated and uploaded, and the system trains itself on the new data.

The intents in column 2 are created as new actions, and the phrases in column 1 are created as example phrases for the corresponding action. For example, if you upload the example from step 1, two new actions are created for the `weather_conditions` and `find_location` intents. The underscores (`_`) in the intent names are replaced with spaces, for example, the `weather_conditions` intent becomes the `weather conditions` action.

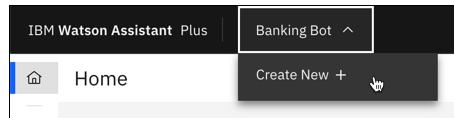
In this example, the `weather_conditions` action will have three example phrases: `Tell me the current weather conditions.`, `Is it raining?`, and `What's the temperature?`. The `find_location` action will have two example phrases: `Where is your nearest location?` and `Do you have a store in Raleigh?`.

## Adding more assistants

You can create multiple assistants in your instance. The number of assistants you can create depends on your Watson Assistant [plan type](#). There is also a limit of 100 assistants per service instance.

If you need to add more assistants, follow these steps:

1. In the top navigation, click the name of your current assistant, then choose **Create New**:



2. Add details about the new assistant:

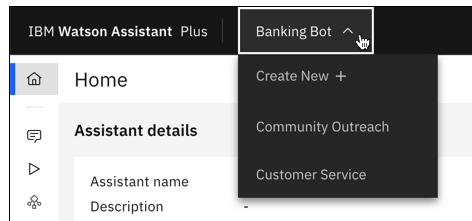
- **Assistant name:** A name no more than 100 characters in length. A name is required.
- **Description:** An optional description no more than 200 characters in length.
- **Assistant language:** The language the assistant uses in conversations. Learn more about [language support](#).

3. Click **Create assistant**.

## **Switching between assistants**

To switch to another assistant in your instance:

1. In the top navigation, click the name of your current assistant.



2. Choose the assistant you want to open.

## **Renaming or deleting assistants**

If necessary, you can rename or delete an assistant. You can't change the language of an assistant once it is set.

### **Renaming an assistant**

You can change the name and description of an assistant after you create it.

To rename an assistant, follow these steps:

1. Switch to the assistant you want to rename.
2. Open **Assistant settings**.
3. In **Details**, enter a new name or description.
4. Click **Save**.



**Note:** You can also access the **Assistant ID** in the **Details** section.

### **Deleting an assistant**

To delete an assistant, follow these steps:

1. Switch to the assistant you want to delete.
2. Open **Assistant settings**.
3. Click the **Delete assistant** button on the settings page. A confirmation question displays.
4. To confirm, type the word **DELETE**, then click **Delete**.

# Previewing and publishing

## Administering your instance

---

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

## Previewing and sharing your assistant

---

Internal review is a necessary step in any virtual assistant workflow. You need an environment without customer interactions so your team can test your assistant. The draft environment closely resembles the final experience that your users encounter so you can ensure that you are publishing the optimal end product.

## Saving and editing your work in the draft environment

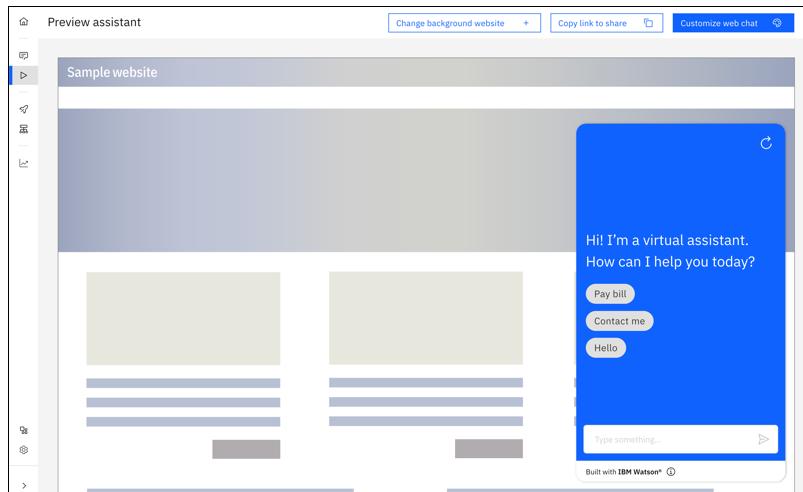
The draft environment contains all your in-progress work in the **Actions**, **Preview**, and **Publish** pages. Use the **Draft environment** tab to manage the draft environment, including adding draft environment integrations (channels and extensions) that you can use for internal testing before deployment. These integrations are unique to the draft environment, and changes to draft integrations don't affect the live environment.

## The Preview page

Use the **Preview** page to test your assistant. You can experience your assistant from your customers' perspective. The **Preview** page includes an interactive web chat widget where you can test out your assistant as if you were a customer. The content that is contained in the assistant is the content that you built into your actions or set up with the search integration.

On the **Preview** page, you also find the following elements:

- **Copy link to share:** Share an unauthenticated version of your assistant with your colleagues by sending them a link. For more information, see [Copying a link to share](#).
- **Change background:** Change the background of the page so you can see what your assistant looks like on different web pages. For more information, see [Changing background website](#).
- **Customize web chat:** Customize your draft web chat channel to match your brand or website. For more information, see [Web chat setup overview](#).



## Copying a link to share

You can share an unauthenticated version of your assistant with your team by sending them a link. The link opens a sample web page with an interactive web chat widget where you can test out your assistant as if you were a customer. Your subject-matter experts can test your in-progress assistant without needing access to Watson Assistant itself. The experience is identical to using **Preview this environment** on the draft environment tab.

To share a link:

1. On the **Preview** page, click **Copy link to share**.
2. Send the link to your team.

**Note:** The preview link is not accessible if web chat security is enabled. For more information about web chat security, see [Securing the web chat](#).

## Changing background website

You can visualize how your assistant would look as a web chat widget on your organization's website. You can enter a URL or upload an image.

### Entering a URL

You can enter a URL of your organization's website. Watson Assistant captures an image of your website to use as the **Preview** page background.

**Note:** Your website must be publicly available to all users. Private or intranet sites can't be accessed. Any login, splash, cookie, or warning screens might be captured in the image.

To enter a URL:

1. On the **Preview** page, click **Change background**.
2. Click **Enter URL**, then click **Continue**.
3. Enter the path of your website URL, for example, <https://www.example.com> or [example.com](http://example.com).
4. Click **Continue**.

## Uploading an image

You can upload an image of your organization's website. Images are stored for 24 hours. Maximum file size is 1 MB. Supported file types are JPEG and PNG.

To upload an image:

1. On the **Preview** page, click **Change background**.
2. Click **Upload an image**, then click **Continue**.
3. Drag a file or click to upload, then click **Change background**.

Images are stored for 24 hours. A warning message might appear on the Preview page about the time limit expiration. To clear this message:

1. On the **Preview** page, click **Change background**.
2. Click **Clear background setting**, then click **Continue**.
3. Click **Remove background** to finish.

## Removing the background

After you enter a URL or upload an image to use as a background, you might decide to remove the background and restore the default background.

To remove the background image:

1. On the **Preview** page, click **Change background**.
2. Click **Remove background**, then click **Continue**.
3. Click **Remove background** to finish.

## Publishing your content

---

Publishing is a way to maintain a healthy lifecycle management process. You can create incremental versions of your content over time, making it easier to manage deployment of changes and roll back (revert) to prior versions if necessary.

When you are ready to create a snapshot of your content and settings, you can publish from the **Publish** page. Each time that you publish, you create a new version, such as V1 or V2.

[Home](#) [Publish](#)

[Unpublished content](#)  
Changes to content made in your draft environment are reflected here.

[Revert](#) [Publish](#)

Draft content	Change type	Content type	Last modified
Book a meeting	Updated	Actions	a minute ago
Check account balance	Updated	Actions	a minute ago
Access customer records	Updated	Actions	a minute ago
Greet customer	Updated	Actions	a minute ago
No action matches	Updated	Actions	2 minutes ago
Fallback	Updated	Actions	2 minutes ago
Trigger word detected	Updated	Actions	2 minutes ago
Actions Settings	Updated	Settings	2 minutes ago

Items per page: 10 ▾ 1-8 of 8 items [1](#) ▾ of 1 page [◀](#) [▶](#)

### [Publish page](#)

When you publish your content, Watson Assistant creates a snapshot of the draft content, resulting in a version. This version contains all of the content from actions, including settings and variables. Versions do not contain integration configurations or environment settings. Integration configurations and environment settings must be configured manually in each environment.

The three most recent published versions appear in a list on the **Publish** page itself. If you have more than three versions, you can click **View all** to see a list of all published versions.

[All versions](#)

<b>V4</b> <a href="#">Live</a> 03/01/2023 11:46AM Contains actions Live version for customers	<b>V3</b> 03/01/2023 11:45AM Contains actions Updated Business Hours action
<b>V2</b> 03/01/2023 11:43AM Contains actions Added Pay Bill action	<b>V1</b> 03/01/2023 11:41AM Contains actions Initial changes

[Close](#)

### [All versions](#)

The number of versions that can be maintained depends on the type of plan you have. If you reach the plan limit of versions you

can have, you need to delete a version before you can publish another one.

Service plan	Published versions
Enterprise	50
Premium (legacy)	50
Plus	10
Trial	10
Lite	2

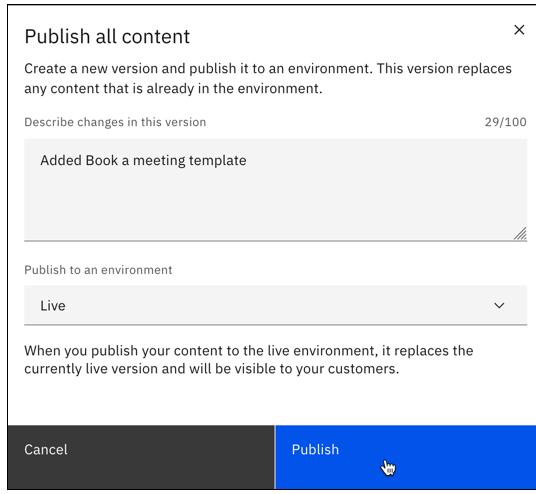
#### Service plan published versions

## How to publish

1. If changes are available to publish, click **Publish**.
2. Enter a description of the version.
3. Choose to publish to the live environment, or to create a version without publishing. If you choose to create a version, you can use the environment tab to publish that version later.

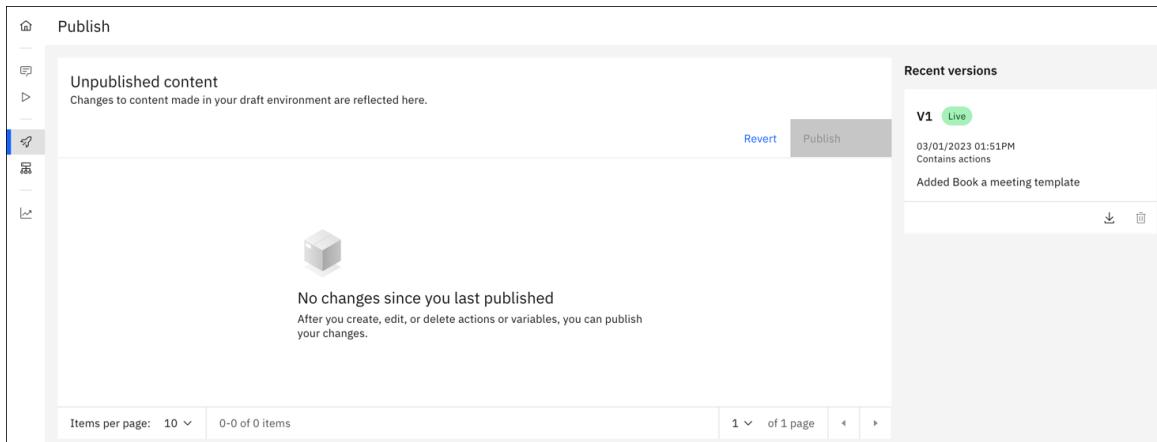
With the Enterprise plan, you can publish to an environment you added. For more information, see [Adding and using multiple environments](#).

4. Click **Publish**:



#### Publish all content

5. After you publish, the list is clear until you make more edits to your content.



No changes

## What is being published?

When you publish, a snapshot is taken of the current state of actions. All updates to actions content are published in a version, including the following updates:

- Creating an action
- Editing an action
- Editing actions settings
- Deleting actions
- Creating variables
- Editing variables
- Deleting variables

If you are using [dialog with actions](#), these updates are published:

- Activating dialog
- Creating a dialog
- Editing a dialog
- Editing dialog settings
- Deleting a dialog

The following updates are not published in a version and must be configured manually for each environment:

- Channel configurations
- Environment settings

## Publishing or switching versions in the live environment

There are two ways to publish a version to the live environment:

- When you publish a version from the draft environment
- Use the **Live environment** tab and click **Publish version**.

After a version is published to the live environment, there are two ways you can switch to a different version:

- When you publish a version from the draft environment, you have the option of assigning it to the live environment, replacing the one that's already there
- Use the **Live environment** tab and click **Switch version**.

## Reverting to a previous version

Use the revert function if you need to update a version of content that is already published to a version. For example, if you publish V1 of your content and then notice an error, you can revert your actions to V1 and use the draft environment to correct the error.

If you revert to a previous version, your published content isn't affected, but the content on the **Actions** page is overwritten with the version that you revert to. For example, if you decide to revert to V1 of your content, all content on the **Actions** page is overwritten with the V1 content so you can continue to work on it.

To revert to a previous version:

1. On the **Publish** page, click **Revert**.
2. On the **Publish content** tab, choose whether you want to save your in-progress actions as a content version so that you don't lose your current work. For example, if you are reverting to V1 and you choose to create a version, V2 is created so you can resume work on it later.

To create a version, check **Publish content** and add a description.

Revert to a prior version

Publish content

Create a new version with draft content

Create a non-live version of your in-progress content so that it is not lost. You can revert to this version later.

Publish content

Version description (optional) 0/100

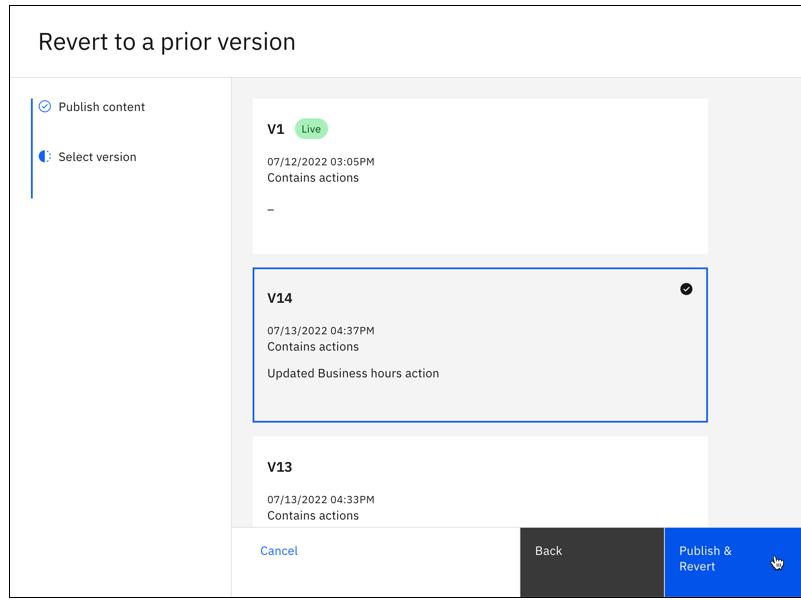
Example: Updated "Business hours" action

Add a description to help identify what content this version contains.

Cancel Back Next

**Publish content**

3. Next, select the version of content that you want to revert to, then click **Publish & Revert**.



### Revert to a prior version

4. After you revert, you can go to the **Actions** page to make any fixes or updates. When you're ready to publish, go to the **Publish** page and publish your updated content as a new version (for example, V3).

## Adding and using multiple environments

### Enterprise

Each assistant has a draft and live environment. For Enterprise plans, you can add up to three environments as a staging area to test your assistant before deployment. You can build content in the draft environment and test versions of your content in the extra environments.

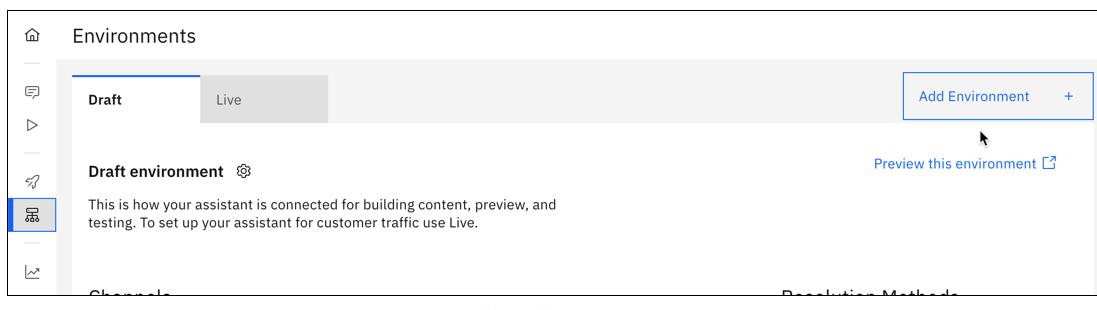
For more information about environments, see [Environments](#).

### Adding environments

Each new environment appears as an extra tab on your **Environments** page. You can't reorder these environments, so add them to match your test-to-deploy lifecycle.

To add an environment:

1. Open the **Environments** page and click **Add Environment**.



### Add environment

2. Enter a name and a description, then click **Save**. Names can't contain spaces or use any special characters.

Add an environment

To enable an additional environment within your assistant, enter a name and description.

This will be your 3rd of 5 environments in this assistant.

Name 7/24  
Staging

Description(optional) 44/100  
Staging area for all Banking Bot deployments

Cancel Save

Add an environment

The new environment appears as an extra tab on your **Environments** page.

Environments

Draft Staging Live

Staging environment ⓘ  
Staging area for all Banking Bot deployments

Environment tab

## Using your environments

You can use extra environments in your development and test process before you deploy an assistant for customer use.

Add extra environments to match your existing test-to-deploy process. For example, you might name and use five environments in a scenario like this:

Environment	Used for
Draft	Conversation authors build actions and test as they work
Review	Conduct an initial content review with stakeholders
Test	Test assistant content on a configured channel
Staging	Use a staging website to test assistant content and channel configurations
Live	Deploy for customer use

Example

## Access control to environments

You can control who can work in each environment. Each environment has an ID that you can use in IBM Cloud Identity and Access Management (IAM) and set access by resource. For more information on access control, see [Managing access with Identity and Access Management](#). For more information on settings, see [Environment settings](#).

## Publishing content to an environment

For more information, see [Publishing your content](#).

To publish new changes from the draft environment to an environment:

1. If changes are available to publish, click **Publish**.
2. Enter a description of the version.
3. Choose an environment.
4. Click **Publish**.

To publish an existing version to an environment:

1. On the **Environments** page, click the environment tab.
2. In Resolution Methods, click **Switch version**.
3. Choose a version to publish, then click **Switch version**.

## Moving a version through multiple environments

This example explains moving a content version through multiple environments to build, test, iterate, and deploy.

Environment	Activity
Draft	Publish version V3 to Review environment
Review	Conduct initial testing of V3
Test	Switch to version V3 for further testing with a configured channel
Staging	Switch to version V3 for testing with an internal staging website
Live	Switch to version V3 for customer use
Live	Switch to version V2 after a bug is found in version V3
Draft	Revert to version V3 to fix the bug. For more information, see <a href="#">Reverting to a previous version</a>
Draft	Publish version V4 to Review environment for retesting
Test	Switch to version V4 for further retesting
Staging	Switch to version V4 for testing with an internal staging website
Live	Switch to version V4 for customer use

### Example

## Previewing an environment

On each environment tab, you can click **Preview this environment** to open another browser tab and preview your assistant as an

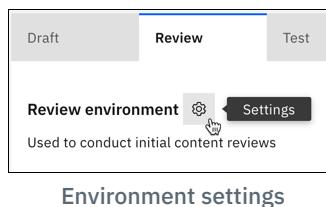
interactive web chat widget.



You can share this unauthenticated version of your assistant with your team by sending them the link to the environment preview. Your subject-matter experts can test your in-progress assistant without needing access to Watson Assistant itself.

## Environment settings

Each environment has its own settings. On an environment tab, click the **Settings** gear icon to open the settings:



**API details** provide these values for each environment:

- Environment name
- Environment ID
- Session URL

### Webhooks

Settings for pre-message, post-message, and log webhooks. For more information, see [Extending your assistant with webhooks](#).

### Inactivity timeout

Specify the amount of time to wait after the customer stops interacting with the assistant before ending the session. The maximum inactivity timeout differs by service instance plan type. For more information, see [Environment settings](#).

### Edit environment

Change the name or description of the environment. Names can't contain spaces or use any special characters.

### Delete environment

If necessary, you can remove the environment. Any channel or extension configurations are removed. Deleting an environment doesn't delete any published content versions. They remain in your list of published versions.

# Deploying to your website or mobile app

## Administering your instance

---

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

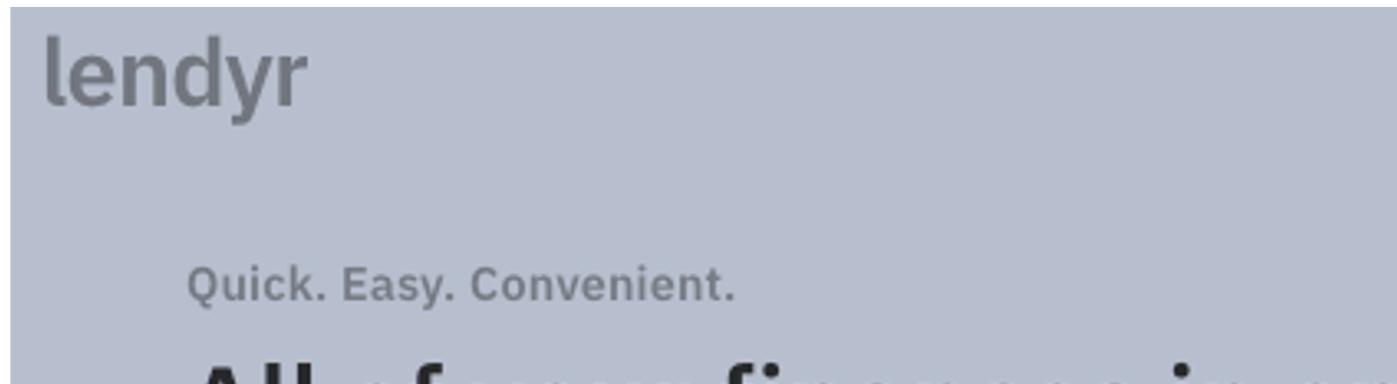
Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

## How the web chat works

---

The web chat provides an easy-to-use chatbot interface that you can add to your website without writing any code.

After you add the web chat script to your website, your customers will see a launcher icon that they can click to open the chat window and start a conversation with the assistant. The appearance of the launcher icon adapts to desktop and mobile browsers.



# All of your finances in one place Designed to work better



Student Loans

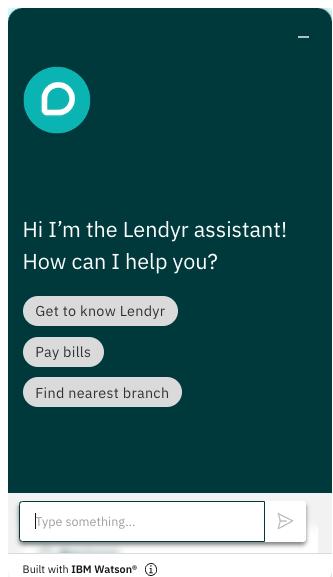


Personal Loans





When a customer clicks the launcher, the web chat window opens, initially displaying the *home screen*. The home screen displays a greeting and an optional set of suggested conversation starters for common questions and problems. The customer can either click a conversation starter or type a message in the input field to start the conversation with the assistant.



- Tip:** The appearance and behavior of the launcher icon, the home screen, and most other aspects of the web chat can be configured and customized to match your website style and branding. For more information, see [Configuring the web chat](#).

## Launcher appearance and behavior

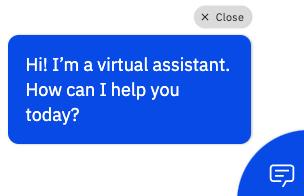
The web chat launcher welcomes and engages customers so they know where to find help if they need it. By default, the web chat launcher appears in a small initial state as a circle in the bottom right corner:



After 15 seconds, the launcher expands to show a greeting message to the user. In this expanded state, a customer can still click the launcher to open the web chat. (If the customer reloads the page or navigates to a different page before the launcher has expanded, the 15-second timer restarts.)

The appearance of this expanded state differs slightly depending on whether the customer is using a desktop browser or a mobile browser:

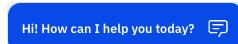
- For desktop browsers, the expanded launcher shows two primary buttons the customer can click to open the web chat, and a **Close** button that closes the launcher.



The expanded launcher remains in its expanded state even if the customer reloads the page or navigates to a different page.

It stays in its expanded state until the customer either opens it by clicking on either of the two primary buttons, or closes it, at which point it returns to its initial small state for the rest of the session.

- For mobile browsers, the launcher shows only a single primary button.



The customer can close the launcher by scrolling on the page, swiping right on the expanded launcher, or waiting 10 seconds, at which point the expanded launcher shrinks back to its initial small state automatically. If the user reloads the page or navigates to a different page while the launcher is expanded, it stays in its expanded state, and the 10-second timer restarts.

After the next page refresh, if the launcher remains in its small state without being clicked, it bounces up and down to attract the customer's attention. The first bounce happens 15 seconds after the page refresh; if the customer still does not click the launcher, it bounces again 60 seconds later. (The timing of the second bounce might be affected if the user refreshes the page or navigates to a different page.) If the user still does not click the launcher, it does not bounce again.

The language of the default text shown within the launcher depends on the locale configured for the web chat. If you customize the greeting text, the text you provide is used regardless of the locale settings.

You can configure the color of the launcher, as well as the greeting message text, in the web chat settings. For more information, see [Configuring the web chat](#).

## Rendering assistant output

In addition to plain text, Watson Assistant supports many response types that can be used to output multimedia and interactive elements. The web chat includes built-in support for a wide variety of response types:

- **Text formatting:** The web chat supports formatting text using either Markdown or HTML. For more information, see [Markdown formatting](#).
- **URLs:** Valid URLs (such as `http://example.com`) are automatically rendered as clickable links. When a customer clicks a link in the web chat, the target website opens in a new browser tab.
- **Options:** Options responses (when the assistant asks the customer to select from a set of choices) are automatically rendered as interactive elements. (By default, a set of fewer than five options is rendered as a set of clickable buttons; five or more options are rendered as a drop-down list.)
- **Dates:** When the assistant asks the customer to specify a date, the web chat displays an interactive date picker. The customer can specify the date either by clicking the date picker or by typing a valid date value in the input field.

- **Multimedia responses:** The web chat supports all multimedia response types `audio`, `image`, and `video`.
- **iframe:** The web chat supports the `iframe` response type, which embeds HTML content (such as a form or interactive map) directly in the web chat window.

For more information about how the web chat handles specific response types, see the [Response types reference](#).

## Markdown formatting

In text responses from your assistant, you can use Markdown formatting to apply highlighting such as italics, or to include elements like paragraphs and headings. Some common examples of Markdown formatting include:

- Headings:

```
$ # First-level heading
## Second-level heading
```

- Highlighting:

```
$ This text includes *italic* and **bold** highlighting, as well as a `code` snippet.
```

- Lists:

```
$ - ordered
  - list
  1. bulleted
  2. list
```

- Tables:

```
$ | Column 1 | Column 2 |
|-----|-----|
| Row    | One    |
| Row    | Two    |
```

- Links:

```
$ [This link](https://www.ibm.com/products/watson-assistant/demos/lendyr/demo.html) opens in a new tab.

[This link](https://www.ibm.com/products/watson-assistant/demos/lendyr/demo.html) {{target=_self" rel="noopener noreferrer"}} opens in the same tab.
```

For detailed information about the Markdown format, see the [CommonMark specification](#).

## Live agent transfer

The web chat supports transferring the customer to a human in situations the assistant can't handle. If you configure one of the supported contact center integrations, the web chat can open a separate chat window in which the customer can communicate with a live agent.

Your assistant can then initiate a transfer in situations when the assistant is unable to handle a customer's requests. (For more information about initiating a transfer, see [Connecting to a live agent](#).)

For information about how to add a contact center integration to the web chat, see [Adding contact center support](#).

## Technical details

The web chat is displayed on your web site by a short JavaScript code snippet, which calls additional JavaScript code that is hosted by IBM Cloud. The hosted code is automatically updated with new features and fixes, so by default you will always have the latest version. (You can optionally [lock to a specific version](#) if you prefer to control upgrades yourself.)

The code snippet that creates the web chat widget includes a configuration object, which you can modify to change the appearance and behavior of the web chat. The configuration object also specifies details that enable the web chat to connect to your assistant. If you are comfortable writing JavaScript code, you can customize the web chat by modifying the code snippet and using the web chat API.

The web chat uses the Watson Assistant v2 stateful API to communicate with the assistant. By default, the session ends and the conversation ends after 5 minutes of inactivity. This means that if a user stops interacting with the assistant, after 5 minutes, any context variable values that were set during the previous conversation are set to null or back to their initial values. You can change the inactivity timeout setting in the assistant settings (if allowed by your plan).

## Browser support

The web chat supports a variety of devices and platforms. As a general rule, if the last two versions of a browser account for more than 1% of all desktop or mobile traffic, the web chat supports that browser.

The following list specifies the minimum required browser software for the web chat (including the two most recent versions, except as noted):

- Google Chrome
- Apple Safari
- Mobile Safari
- Chrome for Android
- Microsoft Edge (Chromium and non-Chromium)
- Mozilla Firefox
- Firefox ESR (most recent ESR only)
- Opera
- Samsung Mobile Browser
- UC Browser for Android
- Mobile Firefox

For optimal results when rendering the web chat on mobile devices, the `<head>` element of your web page must include the following metadata element:

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

## Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

The web chat integration complies with the [Web Content Accessibility 2.1 Level AA](#) standard. It is tested with both screen readers and automated tools on a continual basis.

## Language support

By default, the web chat displays hardcoded labels and messages in English, but support is built in for all of the languages supported by Watson Assistant. You can also choose from a wide selection of locales to customize the display of strings like dates and times for global audiences.

In whichever language you are using, you can also customize the text of any hardcoded strings.

For more information, see [Supporting global audiences](#).

## Security

By default, all messages that are sent between the web chat and the assistant are encrypted using Transport Layer Security (TLS). You can enable the web chat security feature if you need more robust protection.

The web chat embed script that you include on your website contains unique identifiers (such as the integration ID and service instance ID) that enable the web chat to connect with your assistant. These identifiers are not considered secret, and are visible to anyone who has access to your website. Anyone who has these IDs could use them to send messages to your assistant and receive its replies. However, these IDs cannot be used to log in to your account, make any changes to your assistant, or retrieve logs or analytics information about your assistant.

If you are concerned about unauthorized access to your assistant, you can enable the web chat security feature for additional security, such as verifying message origin and authenticating users. Enabling the security feature requires additional development work on your website. For more information, see [Web chat security](#).

## Updating site security policies

If your website uses a Content Security Policy (CSP), you must update it to grant permission to the web chat.

The following table lists the values to add to your CSP.

Property	Additional values
default-src	'self' *.watson.appdomain.cloud fonts.gstatic.com 'unsafe-inline'
connect-src	*.watson.appdomain.cloud

**CSP properties**

The following example shows a complete CSP metadata tag:

```
<meta
  http-equiv="Content-Security-Policy"
  content="default-src 'self' *.watson.appdomain.cloud fonts.gstatic.com 'unsafe-inline';connect-src
  *.watson.appdomain.cloud" />
```

## Allowing elements

If your CSP uses a nonce to add elements such as `<script>` and `<style>` tags to an allowlist, do not use `'unsafe-inline'` to allow all such elements. Instead, provide a nonce value to the web chat widget as a configuration option. The web chat will then set the nonce on any of the `<script>` and `<style>` elements that it generates dynamically.

A CSP that passes a nonce to the web chat widget might look like this:

```
$ <meta
  http-equiv="Content-Security-Policy"
  content="default-src 'self' *.watson.appdomain.cloud fonts.gstatic.com 'nonce-<server generated
  value>';connect-src *.watson.appdomain.cloud"
>
```

You can pass the nonce to the web chat by editing the embed script as follows:

```
window.watsonAssistantChatOptions = {
  integrationID: "YOUR_INTEGRATION_ID",
  region: "YOUR_REGION",
```

```

serviceInstanceId: "YOUR_SERVICE_INSTANCE",
cspNonce: "<server generated value>",
onLoad: function(instance) {
  instance.render();
}
};

```

## Reviewing security

The web chat integration undergoes tests and scans on a regular basis to find and address potential security issues, such as cross-site scripting (XSS) vulnerabilities.

Be sure to run your own security reviews to see how the web chat fits in with your current website structure and policies. The web chat is hosted on your site and can inherit any vulnerabilities that your site has. Only serve content over HTTPS, use a Content Security Policy (CSP), and implement other basic web security precautions.

## Billing

Watson Assistant charges based on the number of unique monthly active users (MAU).

By default, the web chat creates a unique, anonymous ID the first time a new user starts a session. This identifier is stored in a first-party cookie, which remains active for 45 days. If the same user returns to your site and chats with your assistant again while this cookie is still active, the web chat integration recognizes the user and uses the same user ID. This means that you are charged only once per month for the same anonymous user.

**⚠ Important:** On Apple devices, the Intelligent Tracking Prevention feature automatically deletes any client-side cookie after 7 days. This means that if an anonymous customer accesses your website and then visits again two weeks later, the two visits are treated as two different MAUs. For information about how to avoid this problem, see [Managing user identity information](#).

For information about how to customize the handling of user identity information for billing purposes, see [Managing user identity information](#).

The usage is measured differently depending on the plan type. For Lite plans, usage is measured by the number of /message calls (API) are sent to the assistant from the web chat integration. For all other plans, usage is measured by the number of monthly active users (MAU) that the web chat interacts with. The maximum number of allowed MAUs differs depending on your Watson Assistant plan type.

Plan	Maximum usage
Enterprise	Unlimited MAU
Premium (legacy)	Unlimited MAU
Plus	Unlimited MAU
Trial	5,000 MAU
Lite	10,000 API (approximately 1,000 MAU)

[Plan details](#)

## Web chat setup

### Administering your instance

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

### Configuring style and appearance

On the **Style** tab, you can configure the overall appearance of the web chat widget. You can make the following changes:

- Set the assistant name. Click **Assistant's name as known by customers** to specify the name that is displayed in the header of the chat window. The name can be up to 64 characters in length.
- Change the colors used in the web chat widget. You can set the following colors:
  - **Primary color:** The color of the web chat header.
  - **Secondary color:** The color of the customer input message bubble.
  - **Accent color:** The color of interactive elements such as the following:
    - Web chat buttons such as the suggestions button and the "send message" button
    - The border around the input text field (when in focus)
    - The markers that appear beside the assistant's messages
    - The borders that appear around buttons and drop-down lists when customers select from options
    - The home screen background

Each color is specified as an HTML hexadecimal color code, such as `#FF33FC` for pink and `#329A1D` for green. To change a color, type the HTML color code you want to use, or click the dot next to the field and choose the color from the interactive

color picker.

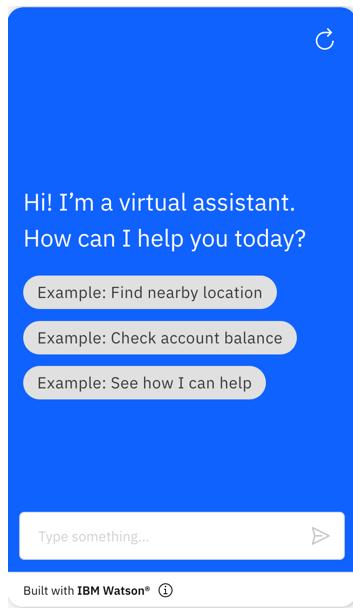
- Enable or disable the **Built with IBM Watson** watermark that is displayed in the web chat widget. To disable the watermark, click to toggle the **IBM Watermark** switch to the off position. (The watermark cannot be disabled on the Lite plan.)
- Provide an avatar image to represent your assistant or organization in the web chat header. Click **Add an avatar image** to add an image, or **Change avatar image** to change an image you have previously added.

Specify the URL for a publicly accessible hosted image, such as a company or brand logo or an assistant avatar. The image file must be between 64 x 64 and 100 x 100 pixels in size.

Style changes you make are immediately reflected by the web chat preview that is shown on the page. However, no configuration changes are applied to the environment until you click **Save and exit**.

## Configuring the home screen

On the **Home screen** tab, you can configure the contents of the home screen, which welcomes customers and helps them start the conversation with the assistant. The home screen replaces any greeting that would otherwise be sent by the *Greet customer* system action.



If you prefer to use a *Greet customer* system action instead of the home screen, you can disable it by clicking the toggle switch on the **Home screen** tab.

If you use the home screen, you must configure it to show content that is relevant to your customers:

- In the **Greeting message** field, type a greeting that is engaging and invites the user to interact with your assistant. This greeting is the first message your customers will see when they open the web chat window.
- In the **Conversation starters** section, specify the conversation starter messages you want to be displayed on the home screen.

These messages are displayed on the home screen as options that customers can click to start the conversation (for example, `I need to reset my password` or `What is my account balance?`). Specify conversation starters that are likely to be useful to your customers, and that your assistant knows how to handle.

Be sure to test each conversation starter. Use only messages that your assistant understands and knows how to answer well. Conversation starters cannot be longer than 35 characters.

You can specify up to 5 conversation starters.

The messages you specify are immediately reflected by the web chat preview that is shown on the page, and you can click the conversation starters to see how your assistant responds. However, no configuration changes are applied to the environment until you click **Save and exit**.

## Configuring suggestions

*Suggestions* give your customers a way to try something else when the current exchange with the assistant isn't delivering what they expect. A question mark icon ? is displayed in the web chat that customers can click at any time to see other topics that might be of interest or, if configured, to request support. Customers can click a suggested topic to submit it as input or click the X icon to close the suggestions list.

If customers select a suggestion and the response is not helpful, they can open the suggestions list again to try a different suggestion. The input generated by the first choice is submitted and recorded as part of the conversation. However, any contextual information that is generated by the initial suggestion is reset when the subsequent suggestion is submitted.

The suggestions are shown automatically in situations where the customer might otherwise become frustrated. For example, if a customer uses different wording to ask the same question multiple times in succession, and the same action is triggered each time, then related topic suggestions are shown in addition to the triggered action's response. The suggestions that are offered give the customer a quick way to get the conversation back on track.

The suggestions list is populated with actions that are relevant in some way to the matched action. The actions are ones that the AI model considered to be possible alternatives, but that didn't meet the high confidence threshold that is required for an action to be listed as a disambiguation option. Any action can be shown as a suggestion, unless its **Ask clarifying question** setting is set to **Off**. For more information about the **Ask clarifying question** setting, see [Asking clarifying questions](#).

To configure suggestions, complete the following steps:

1. Open the **Suggestions** tab.

Suggestions are enabled automatically for new web chat integrations. If you don't want to use suggestions, toggle the switch to **Off**.

2. In the **Include a connection to support** section, specify when you want an option to connect with support to be included in the list of suggestions. You can specify **Always**, **Never**, or **After one failed attempt**.

**After one failed attempt:** Adds the option to the list only if the customer reached a node with an `anything_else` condition in the previous conversation turn or reaches the same action for a second time in succession.

3. In the **Option label** field, type the text of the message that requests help from support. This message is shown as the label for the support option, which is included in the **Suggestions** window under the circumstances you specified in the previous step. If the customer clicks this option, the same message is sent to the assistant.

The message you specify should trigger an action that gives customers a way to connect with support. By default, the message `Connect with agent` is used. If your web chat is integrated with a contact center platform, this message initiates a transfer to a human agent. (For more information about integrating with a contact center, see [Adding contact center support](#).)

If your web chat is not integrated with a contact center, specify a message that helps your customers reach whatever form of support you do offer. If you offer a toll-free support line, you might add `Get the support line phone number`. Or if you

offer an online support request form, you might add [Open a support ticket](#).

Whether you use the default support message or add your own, make sure your action is designed to recognize the message and respond to it appropriately.

## Web chat security

### Administering your instance

---

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

### Enabling web chat security

#### Authenticating users

With web chat security enabled, you can securely authenticate customers by user ID.

The default behavior of the web chat integration is to identify unique users by setting the value of the `user_id` property that is sent as part of each message to the assistant. (For more information, see [Managing user identity information](#).)

This approach is sufficient for tracking unique users for billing purposes, but it is not secure and should not be used for access control. If you enable web chat security, you can use JSON Web Tokens (JWTs) to securely authenticate your users and control access to functions of your assistant that require authorization.

#### Authenticating with the `sub` claim

---



**Note:** To use this method for authenticating users, you must first enable the web chat security feature. For more information, see [Enabling web chat security](#).

When you create a JWT for the web chat, you must specify a value for the `sub` (subject) claim, which identifies the user. (For anonymous users, you can use a generated unique ID.)



**Tip:** When you generate a user ID for an anonymous user, be sure to save the generated ID in a cookie to prevent being billed multiple times for the same customer.

When the web chat integration receives a message signed with this JWT, it stores the user ID from the `sub` claim as `context.global.system.user_id`. For user-based plans, this user ID is used for billing purposes. (You cannot use the `updateUserID()` instance method to set the user ID if web chat security is enabled.) The same user ID is also used as the customer ID, which can be used to make requests to delete user data. Because the customer ID is sent in a header field, the ID you specify must meet the requirements for header fields as defined in [RFC 7230](#).



**Important:** If you are required to comply with GDPR requirements, you might need to persistently store any generated anonymous user IDs, especially for anonymous users who later log in with user credentials. Storing these user IDs makes it possible for you to later delete all data associated with an individual customer if requested to do so.

For more information about user-based billing, see [User-based plans explained](#). For more information about deleting user data, see [Labeling and deleting data](#).

If your customers are required to log in before starting a web chat session, you can use the authenticated user ID as the value of the `sub` claim when you create the JWT. Because the web chat integration validates the JWT and uses the `sub` claim to set the user ID, your assistant can now rely on `context.global.system.user_id` for secure access control to functions that require authorization.



**Important:** After you specify the JWT for the web chat, you cannot change to a JWT with a different `sub` claim during the session. If you need to add authenticated login information in the middle of a session, you can store it as part of the user payload instead. For an example of how to do this, see [Tutorial: Authenticating a user in the middle of a session](#).

## Logging out

To log out a customer, you must destroy the web chat.

If you reload the page when a customer logs out, call the `destroySession()` instance method to remove any reference to the current session from the browser's cookies and storage. If you do not call this method, information that is protected by the JWT is not at risk, but the web chat will try to connect to the previous session and fail.

If you do not perform a full page reload when a customer logs out, call the `destroy()` instance method. The `destroy` method removes the current instance of the web chat that is configured for the current userID from the DOM and browser memory. Next, call the `destroySession()` instance method.

## Encrypting sensitive data

By using the public key that is provided by IBM, you can add an additional level of encryption to hide sensitive data you send from the web chat.



**Note:** To use this method for encrypting data, you must first enable the web chat security feature. For more information,

see [Enabling web chat security](#).

Use this method to send sensitive information in messages that come from your website, such as information about a customer's loyalty level, a user ID, or security tokens to use in webhooks that you call from your actions. Information that is passed to your assistant in this way is stored in a private context variable. (Private variables cannot be seen by customers and are never sent back to the web chat.)

For example, you might start a business process for a VIP customer that is different from the process you start for less important customers. You do not want non-VIPs to know that they are categorized as such, but you must pass this information to your action so it can change the flow of the conversation. To do this, you can pass the customer MVP status as an encrypted variable. This private context variable is available for use by the action, but not by anything else.

- Tip:**  **Tutorial:** For a tutorial that shows an example of using web chat security to authenticate users and protect sensitive data, see [Tutorial: Authenticating a user in the middle of a session](#).

To encrypt sensitive data, follow these steps:

1. On the **Security** tab of the web chat integration settings, copy the public key from the **IBM provided public key** field. (This field is available only if [web chat security is enabled](#).)
2. In the JavaScript function you use to create your JWT, include in the payload a private claim called `user_payload`. Use this claim to contain the sensitive data, encrypted with the IBM public key.

For example, the following code snippet shows a function that accepts a `userID` and `user_payload`. If a `user_payload` is provided, its content is encrypted and signed with the IBM public key. (In this example, the public key is stored in an environment variable.)

```
// Example code snippet to encrypt sensitive data in JWT payload.
const jwt = require('jsonwebtoken');
const RSA = require('node-rsa');

const rsaKey = new RSA(process.env.PUBLIC_IBM_RSA_KEY);

/**
 * Returns a signed JWT. Optionally, also adds an encrypted user payload
 * as stringified JSON in a private claim.
 */
function mockLogin(userID, userPayload) {
  const payload = {
    sub: userID, // Required
    // The exp claim is automatically added by the jsonwebtoken library.
  };
  if (userPayload) {
    // If there is a user payload, encrypt it using the IBM public key
    // and base64 format.
    payload.user_payload = rsaKey.encrypt(userPayload, 'base64');
  }
  const token = jwt.sign(payload, process.env.YOUR_PRIVATE_RSA_KEY, { algorithm: 'RS256', expiresIn: '1h' });
  return token;
}
```

- Tip:** The user payload must be valid JSON data. The web chat uses the default options of the [Node-RSA](#) library, which expects the encryption scheme `pkcs1_oaep` and the encryption encoding `base64`.

3. When the web chat integration receives a message signed with this JWT, the content of the `user_payload` claim is

decrypted and saved as the `context.integrations.chat.private.user_payload` object. Because this is a private variable, it will not be included in logs.

## Web chat development

### Administering your instance

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

### Embedding the web chat on your page

To add the web chat widget to your website, all you need to do is embed a generated script element in your HTML source.

The web chat integration is automatically included for every assistant, and is configured separately for each environment.

To add the web chat to your website, follow these steps:

1. On the  **Integrations** page, find the **Web chat** tile and click **Open**. The **Open web chat** window opens.
2. In the **Environment** field, select the environment you want the web chat widget to connect to. Click **Confirm**.

The **Web chat** page opens, showing the settings for the web chat integration in the selected environment.

 **Tip:** The preview pane shows what the web chat will look like when it is embedded in a web page. If you see a message that starts with, `There is an error`, you probably haven't added any actions to your assistant yet. After

you add an action, you can test the conversation from the preview pane.

3. Click the **Embed** tab.

A code snippet is generated based on the web chat configuration. You (or a web developer) will add this code snippet to the web page where you want the web chat to appear.

This code snippet contains an HTML `script` element. The script calls JavaScript code that is hosted on an IBM site and creates an instance of a widget that communicates with the assistant.

4. Click the  **Copy to clipboard** icon to copy the embed script to the clipboard.

5. Edit the HTML source for the web page where you want the web chat widget to appear. Paste the code snippet into the page. Paste the code as close as possible to the closing `</body>` tag to ensure that your page renders faster.

**⚠ Important:** Do not modify the `integrationID` or `region` property values in the generated embed script.

If you aren't ready to add the web chat to a live website, you can quickly test it using a local HTML file. Use this HTML code as the source for a test page:

```
<html>
  <head></head>
  <body>
    <title>My Test Page</title>
    <p>The body of my page.</p>
    <!-- copied script elements -->
  </body>
</html>
```

Just copy this code into a file with the `.html` extension, and replace the `script` element with the embed script you copied in the previous step.

**⚠ Note:** The identifiers in the embed script (such as `integrationID` `serviceInstanceID`) are not considered secret, and are visible to anyone who has access to your website. For more information, see [Security](#).

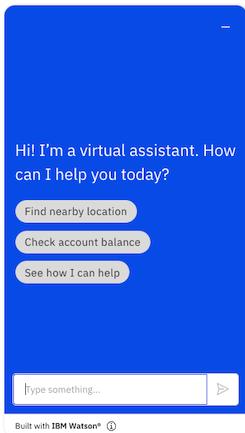
6. If the system that hosts your website has limited Internet access (for example, if you use a proxy or firewall), make sure the following URLs are accessible:

- `https://web-chat.global.assistant.watson.appdomain.cloud`: Hosts the code for the web chat widget, and is referenced by the script you embed on your website.
- `https://integrations.{location}.assistant.watson.appdomain.cloud`: Hosts the web chat server, which handles communication with your assistant. Replace `{location}` with the location of the data center where your service instance is located, which is part of the service endpoint URL. For more information, see [Finding and updating the endpoint URL](#).

7. Open the web page (or local test file) in your browser. You should see the launcher icon displayed on the page:



8. Click the launcher icon to open the chat window.



9. Paste the same embed script into each web page where you want the assistant to be available to your customers.

**Tip:** You can paste the same script tag into as many pages on your website as you want. Add it anywhere where you want users to be able to reach your assistant for help. However, be sure to add it only once on each page.

You can now test your assistant and see its responses just as your customers would.

Before you go to production with the web chat, however, you will probably want to customize it for your site and for the needs of your customers. For more information, see [Web chat development overview](#).

## Adding the web chat to your mobile application

If you have a mobile application built on a mobile framework such as iOS, Android, or React Native, you can use a `WebView` with a JavaScript bridge to communicate between your app and the Watson Assistant web chat.

Using `WebViews` with a JavaScript bridge is a common pattern with a similar implementation for all mobile frameworks.

### Including the web chat as a `WebView`

You can include the web chat interface as part of a page of your mobile app, or as a separate panel that your app opens. In either case, you must host an HTML page that includes the web chat embed script, and then include that page as a `WebView` in your app.

In the embed script, use the `showLauncher` option to hide the web chat launcher icon, and the `openChatByDefault` option to open the web chat automatically when the page loads. In most cases, you will also want to use the `hideCloseButton` option and use the native controls of your app to control how the web chat page or panel closes. For more information about the configuration options you can specify in the embed script, see the [Web chat API reference](#).

The following example shows an embed script that includes these configuration options:

```
$ <html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>
<body>
  <script>
    window.watsonAssistantChatOptions = {
      // A UUID like '1d7e34d5-3952-4b86-90eb-7c7232b9b540' included in the embed code provided in
      Watson Assistant.
      integrationID: "YOUR_INTEGRATION_ID",
      // Your assistants region e.g. 'us-south', 'us-east', 'jp-tok' 'au-syd', 'eu-gb', 'eu-de',
      etc.
```

```

region: "YOUR_REGION",
// A UUID like '6435434b-b3e1-4f70-8eff-7149d43d938b' included in the embed code provided in
Watson Assistant.
serviceInstanceId: "YOUR_SERVICE_INSTANCE_ID",
// The callback function that is called after the widget instance has been created.
onLoad: function(instance) {
  instance.render();
},
showLauncher: false, // Hide the web chat launcher, you will open the WebView from your
mobile application
openChatByDefault: true, // When the web chat WebView is opened, the web chat will already be
open and ready to go.
hideCloseButton: true // And the web chat will not show a close button, instead relying on
the controls to close the WebView
};
setTimeout(function(){const t=document.createElement('script');t.src="https://web-
chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') +
"/WatsonAssistantChatEntry.js";document.head.appendChild(t);});
</script>
</body>
</html>

```

 **Note:** In your app, make sure you include logic to hide your web chat launching mechanism when the device is offline. If the device goes offline in the middle of a conversation, appropriate error messages and retries occur.

## Using a JavaScript bridge

In some situations, your mobile application might need to communicate with the Watson Assistant web chat. For example, you might need to set initial context data (such as a user ID or account information) or use the web chat security feature. To do this, you can use a JavaScript bridge.

A JavaScript bridge is a common pattern that can be used on all mobile platforms. Implementation details differ from one mobile application framework to another; you can find specific examples on how to implement a JavaScript bridge for the framework you are using. However, the core concept of sending events back and forth between the mobile app and the web chat apply to all frameworks.

With a JavaScript bridge, events are sent between the mobile app and the WebView, and event listeners exist on both sides of the bridge to handle those events. Each event carries a message payload; because this payload is text-only, you must stringify and parse JSON objects to pass data back and forth.

If your mobile application needs to call a web chat instance method, you must call the method by sending an event from the app to the WebView using the JavaScript bridge. Similarly, if you need to run code in your mobile application in response to behavior in the web chat, you can send an event through the JavaScript bridge from the web chat to your mobile application.

You can make use of the `user_defined` response type and the `customResponse` event to drive behavior on your mobile application. However, you must strip the `event.data.element` property from the event before you pass it through the JavaScript bridge. Removing this property is necessary because it contains an HTML element, which cannot be stringified. Any use of the `user_defined` response type to write new views into the web chat must be written in HTML and JavaScript and handled inside the WebView. (For more information about the `customResponse` event, see the [Web chat API reference](#).)

## Managing user identity information

Watson Assistant charges based on the number of unique monthly active users (MAU).

If you do not pass an identifier for the user when the session begins, the web chat creates one for you. It creates a first-party cookie with a generated anonymous ID. The cookie remains active for 45 days. If the same user returns to your site later in the

month and chats with your assistant again, the web chat integration recognizes the user. And you are charged only once when the same anonymous user interacts with your assistant multiple times in a single month.

If you want to perform tasks where you need to know the user who submitted the input, then you must pass the user ID to the web chat integration. Choose a non-human-identifiable ID. For example, do not use a person's email address as the user ID.

In addition, the ability to delete any data created by someone who requests to be forgotten requires that a `customer_id` be associated with the user input. When a `user_id` is defined, the product can reuse it to pass a `customer_id` parameter. (For more information about deleting user data, see [Labeling and deleting data](#).)

Because the `user_id` value that you submit is included in the `customer_id` value that is added to the `X-Watson-Metadata` header in each message request, the `user_id` syntax must meet the requirements for header fields as defined in [RFC 7230](#).

To support these user-based capabilities, add the [updateUserID\(\) method](#) in the code snippet before you paste it into your web page.



**Note:** If you enable security, you set the user ID in the JSON Web Token instead. For more information, see [Authenticating users](#).

In the following example, the user ID `L12345` is added to the script.

```
<script>
  window.watsonAssistantChatOptions = {
    integrationID: 'YOUR_INTEGRATION_ID',
    region: 'YOUR_REGION',
    serviceInstanceId: 'YOUR_SERVICE_INSTANCE',
    onLoad: function(instance) {
      instance.updateUserID('L12345');
      instance.render();
    }
  };
  setTimeout(function(){
    const t=document.createElement('script');
    t.src='https://web-chat.global.assistant.dev.watson.appdomain.cloud/versions/' +
      (window.watsonAssistantChatOptions.clientVersion || 'latest') +
      '/WatsonAssistantChatEntry.js';
    document.head.appendChild(t);
  });
</script>
```

## Apple devices

On Apple devices, the Intelligent Tracking Prevention feature automatically deletes any client-side cookie after 7 days. This means that if an anonymous customer accesses your website and then visits again two weeks later, the two visits are treated as two different MAUs.

To avoid this problem, use a server-side first-party cookie in your web application. For example, when an anonymous user visits your website for the first time, you can generate a unique user ID and store it in a server-side cookie with any expiration date you choose. Then, your code can use the [updateUserID\(\) instance method](#) to set the user ID. You can then use the same cookie to set the same user ID for this customer on any future visits until it expires.

## More information

For more information about billing, see [User-based plans explained](#).

For more information about MAU limits per plan, see [Billing](#).

For more information about deleting a user's data, see [Labeling and deleting data](#).

## Supporting global audiences

You can build an assistant that understands customer messages in any of the languages that are supported by the service. The responses from your assistant are defined by you and can be written in any language you want.

However, some of the phrases that are displayed in the web chat widget are part of the web chat itself and do not come from the assistant. By default, these hardcoded phrases are specified in English, but you can apply a different language by adding lines to the embedded web chat script.

The hardcoded phrases used by the web chat widget are specified in *language pack* files. The web chat provides language packs that contain translations of each English-language phrase that is used by the web chat. You can instruct the web chat to use one of these other languages files by using the `instance.updateLanguagePack()` method.

Likewise, the web chat applies an American English locale to content that is added by the web chat unless you specify something else. The locale setting affects how values such as dates and times are formatted.

To configure the web chat for customers outside the US, follow these steps:

1. To apply the appropriate syntax to dates and times *and* to use a translation of the English-language phrases, set the locale. Use the `instance.updateLocale()` method.

For example, if you apply the Spanish locale (`es`), the web chat uses Spanish-language phrases that are listed in the `es.json` file, and uses the `DD-MM-YYYY` format for dates instead of `MM-DD-YYYY`.

 **Note:** The locale you specify for the web chat does not impact the syntax of dates and times that are returned by the underlying skill.

2. To change only the language of the hardcoded English phrases, use the `instance.updateLanguagePack()` method.

For more information, see [Instance methods](#).

3. To change the text direction of the page from right to left, use the `direction` method. For more information, see [Configuration](#).

## Controlling the web chat version

The web chat JavaScript code follows semantic versioning practices. Starting with web chat version 2.0.0, you can set the version of the web chat that you want to use as a configuration option.

If you don't specify a version, the latest version is used automatically (`clientVersion: "latest"`). When you apply the latest version, you benefit from the continuous improvements, feature additions, and bug fixes that are made to the web chat regularly.

However, if you apply extensive customizations to your deployment, such as overriding the theme with your own custom theme, you might want to lock your deployment to a specific version. Locking on a version enables you to test new versions before you apply them to your live web chat.

To use a specific version (`clientVersion: "major.minor.patch"`), specify it as follows:

The following examples show what to specify when the current version is `2.3.1`.

- If you want to stay on a major version, but get the latest minor and patch releases, specify `clientVersion: "2"`.
- If you want to stay on a minor version, but get the latest patch releases, specify `clientVersion: "2.3"`.

- If you want to lock on to a specific minor version and patch release, specify `clientVersion: "2.3.1"`.

To test the updates in a version release of the web chat before you apply the version to your live web chat, follow these steps:

1. Lock the web chat that is running in your production environment to a specific version.
2. Embed your web chat deployment into a new page in a test environment, and then override the version lock setting. For example, specify `clientVersion: "latest"`.
3. Test the web chat, and make adjustments to the configuration if necessary.
4. Update your production deployment to use the latest version and apply any other configuration changes that you determined to be necessary after testing.

For more information about features that were introduced in previous web chat versions, see the [Web chat release notes](#).

## Adding service desk support

### Administering your instance

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

## Integrating with Salesforce

IBM Cloud

Integrate your web chat with a Salesforce service desk solution so your customers always get the help they need.

Integrate with a Salesforce service desk by deploying your assistant with the web chat integration. The web chat serves as the client interface for your assistant. If, in the course of a conversation with your assistant, a customer asks to speak to a person,

you can transfer the conversation directly to a Salesforce agent.

Salesforce is a customer relationship management solution that brings companies and customers together. It is one integrated CRM platform that gives all your departments, including marketing, sales, commerce, and service, a single, shared view of every customer.

## Before you begin

To connect to a Salesforce service desk, your organization must have a Salesforce Service Cloud plan that supports Live Agent Chat. Chat support is available in Salesforce Service Cloud Unlimited and Enterprise plans. It is also available with Performance or Developer plans that were created after 14 June 2012.

Your organization must have a [Salesforce chat app](#) with the following characteristics:

- Console navigation
- Navigation items: Cases, Chat sessions, Chat transcripts
- User profiles: Apply the appropriate profiles to ensure that agents can access the app and view chat history information. You can limit access to this page later. See [Profiles](#).
- A [chat deployment](#).
- A [chat button deployment](#).
- Routing must be configured for the chat button. See [Chat routing options](#).
- If you choose omni-channel routing, be sure to include omni-channel as a utility in the chat app. See [Omni-Channel](#).

You must have a level of access to your Salesforce service desk deployment that allows you to do the following things:

- Edit the chat app
- Get chat deployment and button code details
- Add custom fields to layout objects
- Create Visualforce pages

If you don't, ask someone with the appropriate level of access to perform this procedure for you.

## Setting up the Salesforce service desk connection

To set up a Salesforce service desk integration, complete the following steps:

1. Go to your web chat settings. For more information, see [Integrating the web chat with your website](#).
2. From the web chat integration page in Watson Assistant, set the **Allow transfers to live agents** switch to **On**, and then choose **Salesforce** as the service desk type. Click **Next**.
3. For Watson Assistant to connect to a Salesforce service desk, it needs information about your organization's Salesforce chat deployment and button implementations. Specifically, it needs the API endpoint, organization ID, deployment ID, and button ID. The service can derive the values that it needs from code snippets that you copy and paste to this configuration page.

**Tip:** In a separate browser tab or window, open your Salesforce account settings page. Log in with a user ID that has administrative privileges. You must switch back and forth between your Salesforce and Watson Assistant web chat integration setup pages. It's easier to do so if you have both pages open at once.

- Get the deployment code for your Salesforce Agent Configuration chat deployment.

Go to the Salesforce **Feature Settings>Service>Chat>Deployments** page. Find your organization's deployment. Scroll to the end of the chat deployment configuration page and copy the *Deployment Code* snippet.

- Paste the deployment code snippet into the **Deployment code** field in the Watson Assistant Salesforce configuration page.
- Get the Chat Button code.

Go to the Salesforce **Feature Settings>Service>Chat>Chat Buttons & Invitations** page. Find your organization's button implementation. Scroll to the end of the page, and then copy the *Chat Button Code* snippet.

- Paste the chat button code snippet into the **Chat button code** field in the Watson Assistant Salesforce configuration page, and then click **Next**.

4. Add a chat app that enables the Salesforce agent to see a history of the chat. To do so, create a Visualforce page, and then add a chat app to the page.

5. Add custom fields to the Salesforce chat transcript layout.

 **Note:** This is a one-time task. If the fields already exist for your organization, you can skip this step.

These custom fields are referenced from the Visualforce page code that you will use in the next step.

See [Create Custom Fields](#).

From the Salesforce **Data>Objects and Fields>Object Manager>Chat Transcript>Fields & Relationships** page, create the following custom fields:

- **Token:** Stores a Watson Assistant authentication token that secures the communication between Salesforce and your assistant.
  - **Data Type:** Text Area (Long)
  - **Field Label:** x-watson-assistant-key
  - **Field Length:** Specify the maximum length allowed to ensure it can hold a token that might contain over 100,000 characters.

6. Create a Visualforce page.

Visualforce pages are the mechanism that Salesforce provides for you to customize a live agent's console by adding your own pages to it. A Visualforce page is similar to a standard web page, but it provides ways for you to access, display, and update your organization's data. Pages can be referenced and invoked by using a unique URL, just as HTML pages on a traditional web server can be. See [Create Visualforce Pages](#)

- From the web chat integration page in Watson Assistant, copy the code snippet from the Visualforce page markup field.
- Switch to your Salesforce web page. Search for **Visualforce Pages**. Create a page. Add a label and name to the page. Select the *Available for Lightning Experience, Lightning Communities, and the mobile app* checkbox. Paste the code snippet that you copied in the previous step into the page markup field.

7. Add the Visualforce page that you created to the Salesforce chat app.

To ensure the Salesforce agents can see history of the chat between the customer and your assistant, you must add the page that you created earlier into the console that they use to keep track of their work. See [Create and Configure Lightning Experience Record Pages](#).

- From the Salesforce App Launcher, open the chat app that you created for your agents to talk to customers.
- Open the *Chat Transcripts* object, and then select a transcript page.

- Click the *Setup* icon, and then select *Edit Page*.
- Drag the Visualforce component and drop it into the Chat Transcript Record page layout where you want the chat window to be displayed.
- In the component editor, select the Visualforce page that you created earlier, make any adjustments to the component height that you want, and then click *Save*.

**⚠ Important:** If you do make changes, make sure the height of the Visualforce page is 20 px smaller than the height of the component that you add it to. By default, the height of the component is 300 px and the height of the Visualforce page is 280 px. (The height of the Visualforce page is specified in the `height` attribute of the `iframe` HTML element in the code snippet that you copy and paste.)

- Click *Activation*, and then click the APP, RECORD TYPE, AND PROFILE tab.
- Select the apps to which you want to apply the page layout, and then click *Next*.
- Select the appropriate record type, such as Main, and then click *Next*.
- Select user profiles to give the appropriate set of users access to the page. Limit the group to include only those who you want to be able to view chat history information in the page.
- Click *Next*, and then click *Save*.

8. From the Salesforce configuration page in Watson Assistant, click **Save and exit** to finish setting up the connection.

When you test the service desk integration, make sure there is at least one agent with `Available` status.

Watch a 5-minute video that provides an overview of setting up a connection to a Salesforce service desk:



**View video:** [Salesforce Integration: Watson Assistant](#)

## Adding transfer support to your actions

Update an action to make sure it understands when users request to speak to a person, and can transfer the conversation properly.

## Routing based on browser information

When a customer interacts with the web chat, information about the current web browser session is collected. For example, the URL of the current page is collected. You can use this information to add custom routing rules to your actions. For example, if the customer is on the Products page when a transfer to a human is requested, you might want to route the chat transfer to agents who are experts in your product portfolio. If the customer is on the Returns page, you might want to route the chat transfer to agents who know how to help customers return merchandise.

## Integrating with Zendesk

IBM Cloud

Integrate your web chat with a Zendesk service desk solution so your customers always get the help they need.

Integrate with a Zendesk service desk by deploying your assistant with the web chat integration. The web chat serves as the client interface for your assistant. If, in the course of a conversation with your assistant, a customer asks to speak to a person, you can transfer the conversation directly to a Zendesk agent.

Zendesk Chat lets you help customers in real time, which increases customer satisfaction. And satisfied customers are happier customers. To learn more about this service desk solution, see the [Zendesk website](#).

Zendesk Chat is an add-on to Zendesk Support. Zendesk Support puts all your customer support interactions in one place, so communication is seamless, personal, and efficient, which means more productive agents and satisfied customers.

## Before you begin

1. You must have a Zendesk account. If not, create one.

**⚠ Important:** A Zendesk Chat Enterprise plan is required.

2. Decide whether you want to enable security.

If you choose to enable security in Zendesk, you must collect the name and email address of each user. This information must be passed to the web chat so it can be provided to Zendesk when the conversation is transferred.

## Setting up the Zendesk service desk connection

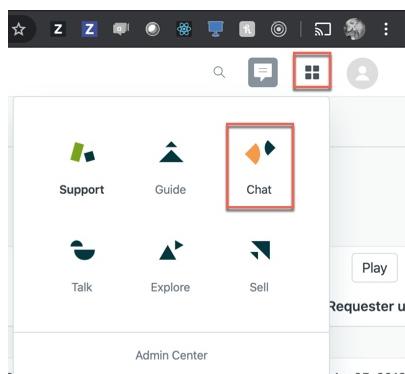
To set up a Zendesk service desk integration, complete the following steps:

1. Go to your web chat settings. For more information, see [Integrating the web chat with your website](#).
2. From the web chat integration page in Watson Assistant, set the **Allow transfers to live agents** switch to **On**, and then choose **Zendesk** as the service desk type, and then click **Next**.
3. Add the account key for your Zendesk account, and then click **Next**.

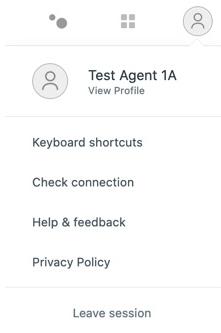
To get the account key for your Zendesk account, follow these steps:

- Log in to your Zendesk subdomain.
- Open the Zendesk Chat Dashboard.

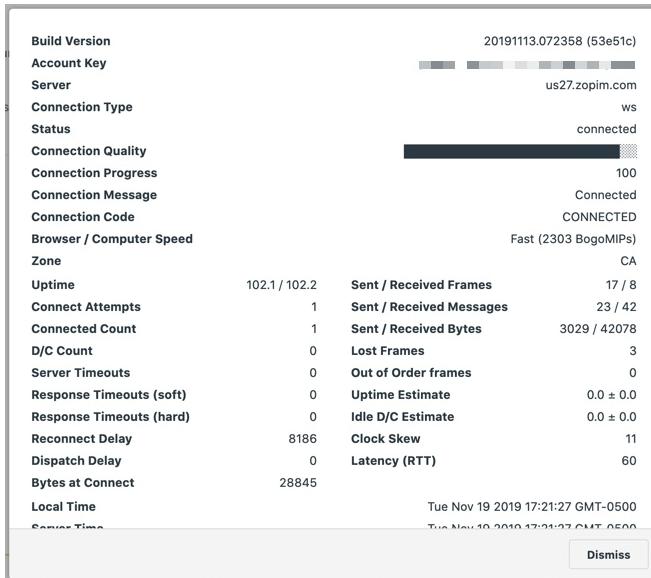
From the Zendesk Support dashboard, you can click the *Zendesk Products* icon in the header, and then click the *Chat* icon.



- Click your profile, and then click *Check Connection*.



- Copy the account key value.



- Return to the setup page in Watson Assistant, and then paste the key into the field.

#### 4. Install the Watson Assistant private application in your Zendesk Chat subdomain.

When you create a Zendesk Chat account, you specify a subdomain. Afterward, your Zendesk console is available from a URL with the syntax: <subdomain>.zendesk.com. For example, ibm.zendesk.com.

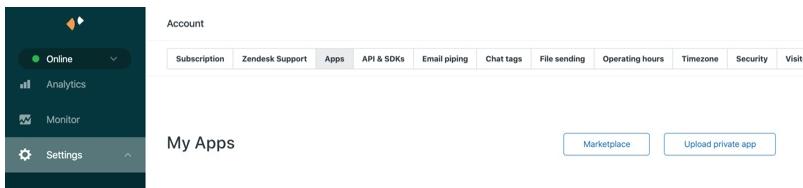
IBM provides an application that you can install in your Zendesk Chat domain. When a customer asks to speak to a person, your assistant will share a chat summary for the transferred conversation with the Zendesk agent by using this private app.

- Download the Watson Assistant Zendesk application from the Zendesk Chat setup page in Watson Assistant.

**Note:** On Safari, the application files are extracted from the ZIP file into a folder. To keep the file archived as a .zip file, so you can upload it later, edit the Safari preferences. Clear the *Open safe files after downloading* checkbox.

- Log in to Zendesk with a user ID that has administrative privileges.
- Install the Watson Assistant Zendesk app to your Zendesk Chat subdomain as a new private app.
  - First, make sure the Zendesk Agent Workspace is not enabled for your account: From the Zendesk navigation pane, go to *Settings*, and then click *Agents*. Deselect the *Enable the Zendesk Agent Workspace* checkbox.
  - Then download the app. From the Chat dashboard navigation pane, expand *Settings*, and then click *Account*.

- Open the App tab.
- Click *Upload private app*, and then browse for the application file that you downloaded earlier.



For more information, see [Uploading and installing a private app in Zendesk Chat](#)

5. Click **Save and exit** to finish setting up the connection to the Zendesk Chat service desk.

When you test the service desk integration, make sure there is at least one agent with **Online** status. Agent status is set to **Invisible** unless it is explicitly changed.

Watch a 4-minute video that provides an overview of setting up a connection to a Zendesk service desk:



[View video: Zendesk Integration: Watson Assistant](#)

**Note:** The product user interface is slightly different from the interface that is shown in the video. However, the main steps are the same.

## Securing the transfer to Zendesk

When you add security to your Zendesk integration, you ensure that the visitors you are helping are legitimate customers. Enabling visitor authentication also enables support for cross-domain traffic and cross-browser identification. For more information, see the [Zendesk documentation](#).

Before you can secure the Zendesk connection, complete the following required tasks:

1. Secure the web chat. For more information see [Securing the web chat](#).
2. Encrypt sensitive information that you pass to the web chat.

When you enable security in Zendesk, you must provide the name and email address of the current user with each request. Configure the web chat to pass this information in the payload.

Specify the information by using the following syntax. Use the exact names `name` and `email` for the two name and value pairs.

```
{
  user_payload : {
    name: '#{customerName}',
    email: '#{customerEmail}'
  }
}
```

For more information, see [Passing sensitive data](#).

Zendesk also expects `iat` and `external_id` name and value pairs. However, there's no need for you to provide this

information. IBM automatically provides a JWT that contains these values.

For example:

```
const userPayload = {  
  "name" : "Cade Jones",  
  "email" : "cade@example.com",  
}  
  
// Sample NodeJS code on your server.  
const jwt = require('jsonwebtoken');  
const RSA = require('node-rsa');  
  
const rsaKey = new RSA(process.env.PUBLIC_IBM_RSA_KEY);  
  
/**  
 * Returns a signed JWT. Optionally, adds an encrypted user_payload in stringified JSON.  
 */  
function mockLogin(userID, userPayload) {  
  const payload = {  
    sub: userID, // Required  
    iss: 'www.ibm.com', // Required  
    acr: 'loa1' // Required  
    // A short-lived exp claim is automatically added by the jsonwebtoken library.  
  };  
  if (userPayload) {  
    // If there is a user payload, it is encrypted in base64 format using the IBM public  
    key.  
    payload.user_payload = rsaKey.encrypt(userPayload, 'base64');  
  }  
  const token = jwt.sign(payload, process.env.YOUR_PRIVATE_RSA_KEY, { algorithm: 'RS256',  
  expiresIn: '10000ms' });  
  return token;  
}
```

3. From the Zendesk application, enable visitor authentication.

- From the Chat dashboard navigation pane, expand *Settings*, and then click *Widget*.
- Open the *Widget security* tab.
- In the *Visitor Authentication* section, click the *Generate* button.

For more information, see [Enabling authenticated visitors in the Chat widget](#). You do not need to follow the steps to create a JWT. The Assistant service generates a JSON Web Token for you.

4. Copy the shared secret from Zendesk.

To secure the Zendesk connection, complete the following steps:

1. In the *Authenticate users* section, set the switch to **On**.
2. Paste the secret that you copied from the Zendesk setup page into the **Zendesk shared secret** field.
3. Decide whether to allow unidentified users to access Zendesk.

The web chat integration allows anonymous users to initiate chats. However, as soon as you enable visitor authentication, Zendesk requires that the name and email of each user be provided. If you try to connect without passing the required information, the connection will be refused.

If you want to allow anonymous users to connect to Zendesk, you can provide fictitious name and email data. Write a function to populate the two fields with fictitious name and email values.

For example, your function must check whether you know the name and email of the current user, and if not, add canned

values for them:

```
const userPayload = {  
  "name" : "Jane Doe1",  
  "email" : "jdoe1@example.com",  
}
```

After writing a function that ensures that name and email values are always provided, set the *Authenticate anonymous user chat transfers* switch to **On**.

## Adding transfer support to your actions

Update an action to make sure it understands when users request to speak to a person, and can transfer the conversation properly.

## Routing based on browser information

When a customer interacts with the web chat, information about the current web browser session is collected. For example, the URL of the current page is collected. You can use this information to add custom routing rules to your actions. For example, if the customer is on the Products page when a transfer to a human is requested, you might want to route the chat transfer to agents who are experts in your product portfolio. If the customer is on the Returns page, you might want to route the chat transfer to agents who know how to help customers return merchandise.

## Tutorials

### Administering your instance

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.

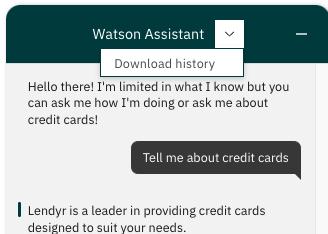
## Tutorial: Providing a downloadable conversation transcript

You can customize the web chat to offer your customers the option of downloading a transcript of the conversation history.



**Note:** For a complete, working version of the example described in this tutorial, see [Download history for Watson Assistant web chat](#).

To support downloading a conversation transcript, this example adds a custom menu option to the overflow menu in the header of the chat window:



Clicking this menu option initiates downloading of a file containing the complete conversation history in comma-separated values (CSV) format.

1. Create a handler for the `send` and `receive` events. In this handler, save each incoming or outgoing message in a list (`messages`) in order to maintain a history of the conversation.

```
$ const messages = [];  
  
function saveMessage(event) {  
  messages.push(event.data);  
}
```

2. Create a handler for the `history:begin` event, which is fired when the web chat is reloaded from the session history. In this handler, save any reloaded session history to the list.

```
$ function saveHistory(event) {  
  messages.push(...event.messages);  
}
```

3. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `send`, `receive`, and `history:begin` events, registering the appropriate handlers as callbacks.

```
$ instance.on({ type: 'send', handler: saveMessage });  
instance.on({ type: 'receive', handler: saveMessage });  
instance.on({ type: 'history:begin', handler: saveHistory });
```

4. Create a function that converts the messages saved in the `messages` list to the format you want to provide in the downloaded file. Note that this conversion needs to accommodate any response types that the conversation might include (such as text, images, options, or transfers to a human agent).

In this example, we convert the messages into a CSV file format that can be opened with an application such as Microsoft Excel. The first column in each line is a label that indicates whether the message originated from the customer (`You`) or from the assistant (`Lendy`).

**Note:** This function relies on a helper function (`createDownloadText`) that formats the text for each line. You can see the implementation of this helper function in the [full example](#).

```
$ function createDownload() {
  const downloadLines = [createDownloadText('From', 'Message')];

  messages.forEach(message => {
    if (message.input?.text) {
      // This is a message that came from the user.
      downloadLines.push(createDownloadText('You', message.input.text));
    } else if (message.output?.generic?.length) {
      // This is a message that came from the assistant. It can contain an array of individual
      message items.
      message.output?.generic.forEach(messageItem => {
        // This is only handling a text response but you can handle other types of responses
        here as well as
        // custom responses.
        if (messageItem?.text) {
          downloadLines.push(createDownloadText('Lendyr', messageItem.text));
        }
      });
    }
  });

  return downloadLines.join('\n');
}
```

5. Create a function that initiates the download of the conversation history file. This function calls the `createDownload()` function to generate the content to download. It then simulates clicking a link to start the download, using a file name generated from the current date.

```
$ function doDownload() {
  const downloadContent = createDownload();

  const blob = new Blob([downloadContent], { type: 'text/csv' });
  const url = URL.createObjectURL(blob);

  // To automatically trigger a download, we have to create a fake "a" element and then click
  it.
  const timestamp = new Date().toISOString().replace(/[_:]/g, '-').replace(/.[0-9] [0-9]Z/, '');
  const a = document.createElement('a');
  a.setAttribute('href', url);
  a.setAttribute('download', `Chat History ${timestamp}.csv`);
  a.click();
}
```

6. In your `onLoad` event handler, use the `updateCustomMenuOptions()` instance method to add a custom menu option that customers can use to download the conversation history. Add this line immediately before the call to the `render()` instance method.

```
$ instance.updateCustomMenuOptions('bot', [{ text: 'Download history', handler: doDownload }]);
```

For complete working code, see the [Download history for Watson Assistant web chat](#) example.

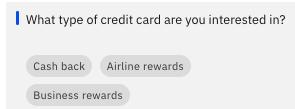
## Tutorial: implementing custom option buttons in the web chat

This tutorial shows how you might replace the default rendering of an options response with your own custom clickable buttons.

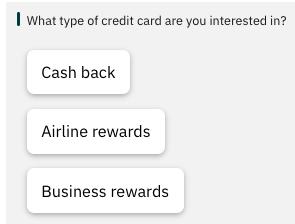
**Note:** For a complete, working version of the example described in this tutorial, see [Custom buttons for Watson Assistant](#)

## [web chat](#)

By default, the web chat always displays an options response as a set of clickable buttons (for 4 or fewer options) or as a drop-down list (for 5 or more options). This example shows the default rendering of an options response with 3 options:



For this tutorial, we will replace this default rendering with larger, card-style buttons:



Because the rendering of an options response cannot be modified, we will do this by intercepting any incoming options responses from the assistant and converting them into custom (`user_defined`) responses. We can then implement a custom rendering for these responses.

1. Create a handler for the `pre:receive` event. In this handler, look for any `option` responses in the message payload, and convert them into `user_defined` responses.

```
$ function preReceiveHandler(event) {
  const message = event.data;
  if (message.output.generic) {
    message.output.generic.forEach(messageItem => {
      if (messageItem.response_type === 'option') {
        messageItem.response_type = 'user_defined';
      }
    })
  }
}
```

2. Create a handler for the `customResponse` event. This handler renders the custom buttons, using a custom `CardButton` style we can define in the CSS. (You can see the definition of this style in the [full example](#).)

```
$ function customResponseHandler(event) {
  const { message, element, fullMessage } = event.data;
  message.options.forEach(messageItem, index) => {
    const button = document.createElement('button');
    button.innerHTML = messageItem.label;
    button.classList.add('CardButton');
    button.addEventListener('click', () => onClick(messageItem, button, fullMessage, index));
    element.appendChild(button);
  };
}
```

3. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `pre:receive` and `customResponse` events, registering the handlers as callbacks.

```
$ instance.on({ type: 'customResponse', handler: customResponseHandler });
instance.on({ type: 'pre:receive', handler: preReceiveHandler });
```

4. Create a click handler to respond when the customer clicks on one of the custom buttons. In the handler, use the `send()`

instance method to send a message to the assistant, using the button label as the message text.

In addition, we're adding the custom CSS class `CardButton--selected` to the clicked button, changing its appearance to show that it was selected. (This class is also defined in the [full example](#).)

```
$ function onClick(messageItem, button, fullMessage, itemIndex) {  
  webChatInstance.send({ input: { text: messageItem.label } });  
  button.classList.add('CardButton--selected');  
}
```

5. If the user reloads the page or navigates to a different page, the web chat reloads from the session history. If this happens, we want to preserve the "selected" state of any clicked buttons.

To do this, in the `onClick` handler, use the `updateHistoryUserDefined` instance method to store a variable in the session history that indicates which button was clicked.

```
$ webChatInstance.updateHistoryUserDefined(fullMessage.id, { selectedIndex: itemIndex });
```

Then, in the `customResponse` handler, read this value and use it to set the initial states of the buttons in any custom responses already in the session history.

```
$ if (fullMessage.history?.user_defined?.selectedIndex === index) {  
  button.classList.add('CardButton--selected');  
}
```

For complete working code, see the [Custom buttons for Watson Assistant web chat](#) example. The example also shows how to disable the buttons in a custom response after the customer has sent a message, which prevents using the buttons to send a message out of order.

## Tutorial: Rendering a custom response as a content carousel

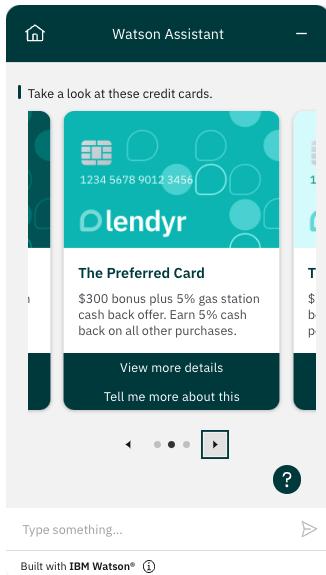
This tutorial shows how you might use custom responses to render information in the form of a content carousel.



**Note:** For a complete, working version of the example described in this tutorial, see [Content carousel for Watson Assistant web chat](#).

A *content carousel* (or *slider*) is a type of interactive element that shows options as a scrollable series of slides.

Watson Assistant does not have a built-in response type for content carousels. Instead, you can use the `user_defined` response type to send a custom response with the information you want to show, and extend the web chat to render the content carousel using standard JavaScript libraries.



This example shows how you can use the [Swiper](#) library to render a custom response as a content carousel.

1. In the action step that you want to create a content carousel, use the JSON editor to define a `user_defined` custom response, which can contain any data you want to include. Inside the response, specify the data required for populating the content carousel. In this example, we're sending information about various types of credit cards, which we will display in a custom response:

```
$ [
  "generic": [
    {
      "user_defined": {
        "carousel_data": [
          {
            "alt": "everyday card",
            "url": "lendyr-everyday-card.jpg",
            "title": "The Everyday Card",
            "description": "$300 bonus plus 5% gas station cash back offer. Earn 2% cash back on all other purchases."
          },
          {
            "alt": "preferred card",
            "url": "lendyr-preferred-card.jpg",
            "title": "The Preferred Card",
            "description": "$300 bonus plus 5% gas station cash back offer. Earn 5% cash back on all other purchases."
          },
          {
            "alt": "topaz card",
            "url": "lendyr-topaz-card.jpg",
            "title": "The Topaz Card",
            "description": "$90 Annual fee. Earn 120,000 bonus points. Earn additional points on every purchase."
          }
        ],
        "user_defined_type": "carousel"
      },
      "response_type": "user_defined"
    }
  ]
]
```

**Note:** In this example, we are including the carousel data inside the `user_defined` response. Depending on the

design of your assistant, another option would be to store the data in skill variables that can be accessed by web chat from the `context` object.

2. Create a handler for the `customResponse` event. This handler renders the content carousel, using the styles defined by the Swiper library. (You can see the definitions of these styles in the [full example](#).)

This function also relies on a helper function (`createSlides()`), which we will create in the next step. (The complete code for this function also initializes the Swiper library; for more information, see the [full example](#).)

```
$ function carouselCustomResponseHandler(event, instance) {
  const { element, message } = event.data;

  element.innerHTML = `
    <div class="Carousel">
      <div class="swiper">
        <div class="swiper-wrapper"></div>
      </div>
      <div class="Carousel__Navigation" >
        <button type="button" class="Carousel__NavigationButton Carousel__NavigationPrevious bx--btn bx--btn--ghost">
          <svg fill="currentColor" width="16" height="16" viewBox="0 0 32 32" aria-hidden="true"><path d="M20 24L10 16 20 8z"></path></svg>
        </button>
        <div class="Carousel__BulletContainer"></div>
        <button type="button" class="Carousel__NavigationButton Carousel__NavigationNext bx--btn bx--btn--ghost">
          <svg fill="currentColor" width="16" height="16" viewBox="0 0 32 32" aria-hidden="true"><path d="M12 8L22 16 12 24z"></path></svg>
        </button>
      </div>
    </div>`;

    // Once we have the main HTML, create each of the individual slides that will appear in the carousel.
    const slidesContainer = element.querySelector('.swiper-wrapper');
    createSlides(slidesContainer, message, instance);

    ...
}
```

3. Create the `createSlides()` helper function that renders each slide in the content carousel. This function retrieves the values from the custom response (`alt`, `url`, `title`, and `description`), and uses them to populate the HTML for the slide. (For the complete function, see the [full example](#).)

```
$ carouselData.forEach((cardData) => {
  const { url, title, description, alt } = cardData;
  const cardElement = document.createElement('div');
  cardElement.classList.add('swiper-slide');

  cardElement.innerHTML =
    <div class="bx--tile Carousel__Card">
      
      <div class="Carousel__CardText">
        <div class="Carousel__CardTitle">${title}</div>
        <div class="Carousel__CardDescription">${description}</div>
      </div>
      <!-- Here you would use a link to your own page that shows more details about this card. -->
      <a href="https://www.ibm.com" class="Carousel__CardButton bx--btn bx--btn--primary" target="_blank">
        View more details
      </a>
      <!-- This button will send a message to the assisstant and web chat will respond with more

```

```

info. -->
  <button type="button" class="Carousel__CardButton Carousel__CardButtonMessage bx--btn bx--btn--primary">
    Tell me more about this
  </button>
</div>
`;
...
});

);

```

4. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `customResponse` event, registering the `carouselCustomResponseHandler()` function as the callback.

```

$ instance.on({
  type: 'customResponse',
  handler: (event, instance) => {
    if (event.data.message.user_defined && event.data.message.user_defined.user_defined_type
    === 'carousel') {
      carouselCustomResponseHandler(event, instance);
    }
  },
});

```

**Note:** In this example, we are checking the custom `user_defined_type` property of the custom response, and calling the `carouselCustomResponseHandler()` function only if the specified type is `carousel`. This is an optional check that shows how you might use a custom property to define multiple different custom responses (each with a different value for `user_defined_type`).

For complete working code, see the [Content carousel for Watson Assistant web chat](#) example.

## Tutorial: Setting context variables for actions from the web chat

This tutorial shows how you can use the web chat to set the values of context variables that your actions can access.

**Note:** For a complete, working version of the example described in this tutorial, see [Setting context variables for Watson Assistant web chat](#).

You can use context variables to store information that the assistant uses throughout the conversation.

Context data is maintained throughout the session. The context is sent to the assistant as part of each message, and returned with each response, in an object called `context`. Context variables for actions skills are stored in the following location:

```

$ "context": {
  "skills": {
    "action skill": {
      "skill_variables": {
        ...
      }
    }
  }
}

```

Any JSON data can be stored in the `skill_variables` object and read or modified by either the actions or the web chat.

This example shows how you can store the customer's name in a context variable, which you can then use to show a personalized greeting. You can use this same approach to store any other information that your assistant might use. Examples

might include the customer's location, an account balance, or stored preferences.

To set the value of a context variable from the web chat, follow these steps:

1. Create a handler for the `pre:send` event. This handler modifies the payload of outgoing message events to assign a value to a context variable called `User_Name`.

We want to set the user name only once at the beginning of the conversation. This example assumes that the home screen is not enabled, which means that the web chat initiates each new conversation with an empty message to trigger a greeting from the assistant.

For this example, we are using the hardcoded user name `Cade`. In a production assistant, you might retrieve the customer's name from a user profile on your website.

```
$ function preSendHandler(event) {
  // Only do this on messages that request the welcome message.
  if (event.data.input && event.data.input.text === '') {
    event.data.context.skills['actions skill'] = event.data.context.skills['actions skill'] ||
  {};
  event.data.context.skills['actions skill'].skill_variables =
  event.data.context.skills['actions skill'].skill_variables || {};
  event.data.context.skills['actions skill'].skill_variables.User_Name = 'Cade';
  }
}
```

2. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `pre:send` event, registering the `preSendHandler()` function as the callback.

```
$ instance.on({ type: 'pre:send', handler: preSendHandler });
```

The assistant actions can now access the `User_Name` variable throughout the conversation.

For complete working code see [the example in our GitHub repository](#).

## Tutorial: Interacting with the host web page

You can use custom responses and events to enable the web chat to interact with the web page where it appears.

For example, your customers might use your assistant to find information they need to complete a form. Rather than expecting the customer to then copy this information manually to the form, you can have the web chat automatically fill in the information.



**Note:** For a complete, working version of the example described in this tutorial, see [Page interactions for Watson Assistant web chat](#).

This example uses a custom response to render a button in the web chat that populates a form field with the customer's account number:

1. Create a handler for the `customResponse` event. This handler renders a custom button and creates a click handler for it. The click handler uses the `Document.querySelector()` method to interact with the DOM and fill in a form field with the customer's account number.

**Note:** This example uses the hardcoded account number `1234567`. In a typical production assistant, your assistant would retrieve this value from a session variable or query it from an external system.

```
$ function customResponseHandler(event) {
```

```

const { element } = event.data;

const button = document.createElement('button');
button.type = 'button';
button.innerHTML = 'Fill in account number';
button.addEventListener('click', () => {
  // Look for the account number element in the document and fill in the account number.
  document.querySelector('#account-number').value = '1234567';
});

element.appendChild(button);
}

```

2. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `customResponse` event, registering the handler as the callback. This enables the assistant to send a custom response that displays the button for filling in the account number.

```

$ instance.on({
  type: 'customResponse',
  handler: (event, instance) => {
    const { message } = event.data;
    if (message.user_defined &&
        message.user_defined.user_defined_type === 'fill_account_number') {
      accountNumberResponseHandler(event, instance);
    }
  },
});

```

**Note:** In this example, we are checking the custom `user_defined_type` property of the custom response, and calling the `accountNumberResponseHandler()` function only if the specified type is `fill_account_number`. This is an optional check that shows how you might use a custom property to define multiple different custom responses (each with a different value for `user_defined_type`).

For complete working code, see the [Page interactions for Watson Assistant web chat](#) example.

## Tutorial: Authenticating a user in the middle of a session

If you have web chat security enabled, you must set a user ID for the customer at the beginning of the session as part of the JSON Web Token (JWT) you use to sign messages. For users who are not authenticated, this is typically a generated ID, and it cannot be changed after the JWT is created. However, you can use a private variable to authenticate a user later in the session.

With web chat security enabled, the user ID associated with each message is based on the `sub` claim in the JWT payload. This value must be set at the beginning of the session, when the JWT is created, and cannot be changed during the life of the session. For unauthenticated (anonymous) users, you would typically use a generated ID, saved to a cookie, to ensure that each unique customer is counted only once for billing purposes.

However, you might want your customers to be able to authenticate in the middle of a session (for example, to complete an action that updates the user's account information). Because the generated user ID in the `sub` claim cannot be changed, you need another way to authenticate the user securely. You can do this by storing the customer's actual authenticated user ID as a private variable in the user payload of the JWT. (You could store the user ID in an ordinary context variable, but this would not be secure, because such variables can be modified.)

**Note:** For a complete, working version of the example described in this tutorial, see [Enabling security for Watson Assistant web chat](#).

This example in this tutorial, which is based on an Express server for Node.js, shows how to start a session with an anonymous

user ID and then authenticate the user during the session.

1. Create a function called `getOrSetAnonymousID()` that generates a unique anonymous user ID for each customer and stores it in a cookie (or, if the cookie already exists, uses the stored user ID).

**Tip:** Use a cookie that lasts for at least 45 days. If you do not store the user ID for more than 30 days, the same customer might be counted as multiple different users during the same billing period. (This can still happen if the same user deletes the cookie or uses a different browser.)

```
$ function getOrSetAnonymousID(request, response) {  
  let anonymousID = request.cookies['ANONYMOUS-USER-ID'];  
  if (!anonymousID) {  
    anonymousID = `anon-${uuid()}`;  
  }  
  
  response.cookie('ANONYMOUS-USER-ID', anonymousID, {  
    expires: new Date(Date.now() + 1000 * 60 * 60 * 24 * 45), // 45 days.  
    httpOnly: true,  
  });  
  
  return anonymousID;  
}
```

1. In the function you use to create a JWT, use the anonymous ID returned from the `getOrSetAnonymousID()` function as the value of the `sub` claim. This sets the value of the user ID that will be used to uniquely identify the customer for billing purposes.

In addition, retrieve any value from the `SESSION_INFO` cookie, which we will use to store authenticated login information. If a value exists, store it in the `user_payload` private claim of the JWT. (If the user has not yet authenticated, this cookie does not yet exist.)

```
$ const jwtContent = {  
  sub: anonymousUserID,  
  user_payload: {  
    name: 'Anonymous',  
    custom_user_id: anonymousUserID,  
  },  
};  
  
if (sessionInfo) {  
  jwtContent.user_payload.name = sessionInfo.userName;  
  jwtContent.user_payload.custom_user_id = sessionInfo.customUserID;  
}
```

1. Create a function to handle user authentication. In our example, we are using a `simpleauthenticate()` function that sets a hardcoded user ID, but in a real application the user ID would probably be retrieved from a database after a secure authentication check. Store the user information in the `SESSION_INFO` cookie.

```
$ function authenticate(request, response) {  
  
  const userInfo = {  
    userName: 'Cade',  
    customUserID: 'cade-id',  
  };  
  
  response.cookie('SESSION_INFO', JSON.stringify(userInfo), { encode: String });  
  response.send('OK');  
}
```

1. When a user logs in, call the `authenticate()` function to store the user information in the `SESSION_INFO` cookie. Then

call the `createJWT()` function to regenerate the JWT, using the updated session info to populate the `user_payload` claim.

In [our example](#), authentication is simulated with a simple button click. The same button also simulates logging out by deleting the cookie:

```
$ async function onClick() {
  if (getCookieValue('SESSION_INFO')) {
    document.cookie = 'SESSION_INFO=; Max-Age=0';
  } else {
    await fetch('http://localhost:3001/authenticate');
  }

  const result = await fetch('http://localhost:3001/createJWT');
  const newToken = await result.text();

  webChatInstance.updateIdentityToken(newToken);

  updateUI();
}
```

The anonymous ID in the `sub` claim will continue to be used to track the customer for billing purposes, but now the customer's real user ID is stored separately in the user payload.

2. In your actions, you can now access the customer's real user ID by referencing the `user_payload` private context variable:

```
$ ${system_integrations.chat.private.user_payload}.custom_user_id
```

For complete working code, see the [Enabling security for Watson Assistant web chat](#) example.

 **Important:** If you are required to comply with GDPR requirements, you might need to persistently store any generated anonymous user IDs, especially for anonymous users who later log in with user credentials. Storing these user IDs makes it possible for you to later delete all data associated with an individual customer if requested to do so.

## Tutorial: Customizing the size and position of the web chat

This tutorial shows how you can change the size and position of the web chat by rendering it in a custom element.

 **Note:** For a complete, working version of the example described in this tutorial, see [Custom elements for Watson Assistant web chat](#).

By default, the web chat interface on your website is rendered in a host `div` element that is styled to appear in a fixed location on the page. If you want to change the size or position of the web chat, you can specify a custom element as the host location for the web chat. This host element is used as the location for both the main web chat interface and for the web chat launcher (unless you are using a custom launcher).

When you use a custom element, you also take control of showing and hiding the web chat when it is opened or closed (such as when the customer clicks the launcher icon or the minimize button). This gives you the opportunity to apply additional effects, such as opening and closing animations. You can control showing and hiding the main window by using the [addClassName\(\)](#) and [removeClassName\(\)](#) functions.

To use a custom element, follow these steps:

1. In your website code, define the custom element where you want the web chat to be rendered. There are many ways of

doing this, depending on the framework you are using. A simple example is to define an empty HTML element with an ID:

```
$ <div id="WebChatContainer"></div>
```

2. Get a reference to your custom element so you can reference it in the web chat configuration. To get a reference, use whatever mechanism makes sense for the library you are using. For example, you can save the reference returned from `document.createElement()`, or you can use a query function to look up the element in the DOM. This example looks up the element using the ID we assigned to it:

```
$ const customElement = document.querySelector('#WebChatContainer');
```

3. In the web chat embed script, set the `element` property, specifying the reference to your custom element.

```
$ window.watsonAssistantChatOptions = {  
  integrationID: "YOUR_INTEGRATION_ID",  
  region: "YOUR_REGION",  
  serviceInstanceId: "YOUR_SERVICE_INSTANCE_ID",  
  
  // The important piece.  
  element: customElement,  
  
  onLoad: function(instance) {  
    instance.render();  
  }  
};
```

4. Make sure that the main web chat window is hidden by default. You can do this in the `onLoad` event handler, after `render` has been called. You must also add handlers to listen for the `window:open` and `window:close` events so the customer can open and close the web chat after the page loads. In our example, we are using a CSS class called `HideWebChat` to do this (see the [full example](#) for the definition of this class):

```
$ function onLoad(instance) {  
  instance.render();  
  instance.on({ type: 'window:close', handler: closeHandler });  
  instance.on({ type: 'window:open', handler: openHandler });  
  instance.elements.getMainWindow().addClassName('HideWebChat');  
}
```

5. Create handlers to hide and show the main web chat window in response to the `window:open` and `window:close`. This example simply shows and hides the element; the [full example](#) shows how you can add animation effects.

```
$ function closeHandler(event, instance) {  
  instance.elements.getMainWindow().addClassName('HideWebChat');  
}  
  
function openHandler(event, instance) {  
  instance.elements.getMainWindow().removeClassName('HideWebChat');  
}
```

For complete working code, see the [Custom elements for Watson Assistant web chat](#) example.

## Tutorial: Displaying a pre-chat or post-chat form

This tutorial shows how you can display pre-chat form before the web chat opens, or a post-chat form that opens after the web chat closes.



**Note:** For a complete, working version of the example described in this tutorial, see [Pre and post-chat forms for Watson Assistant web chat](#).

If you want to gather information from your customers before starting a chat session, you can display a pre-chat form before opening the web chat. Similarly, you might want to display a form after the web chat closes (for example, a customer satisfaction survey). You can use the same approach for either situation.

When the web chat is opened or closed, it fires an [event](#) that you can subscribe to. In your event handler, you can use the [custom panels](#) feature to open a panel with custom content.

By returning a promise that is resolved when the custom panel closes, you can pause the process of opening or closing the web chat until after the customer completes the form.

**Tip:** This example shows how to create a pre-chat form. To create a post-chat form, follow the same steps, but subscribe to the `window:pre:close` event instead of the `window:open` event.

To display a pre-chat form, follow these steps:

1. Create a handler for the `window:open` event, which is fired when the web chat opens. This handler uses the [customPanels.getPanel\(\)](#) instance method to open a custom panel that will contain the pre-chat form.

**Tip:** Your handler should return a promise that is resolved when the custom panel is closed. This prevents the web chat window from opening until after the pre-chat form is completed.

```
function windowOpenHandler(event, instance) {
  return new Promise((resolve) => {
    // Save a reference to the resolve function so we can resolve
    // this promise later.
    promiseResolve = resolve;
    createOpenPanel(event, instance);

    const customPanel = instance.customPanels.getPanel();
    customPanel.open({ hidePanelHeader: true,
                      disableAnimation: true });
  });
}
```

2. In your `onLoad` event handler, use the [on\(\)](#) instance method to subscribe to the `window:open` event, registering the handler as the callback.

```
instance.on({ type: 'window:open', handler: windowOpenHandler });
```

3. Create a function that creates the pre-chat form you want to show inside the custom panel. Make sure you resolve the promise when the user closes the panel.

```
function createOpenPanel(event, instance) {
  const customPanel = instance.customPanels.getPanel();
  const { hostElement } = customPanel;

  hostElement.innerHTML = `
    <div class="PreChatForm">
      ... // Content of pre-chat form
    </div>
  `;

  const okButton = hostElement.querySelector('#PreChatForm__OkButton')
  okButton.addEventListener('click', () => {
    customPanel.close();
    promiseResolve();
  });
}
```

}

- Tip:** A [React version](#) of this example is also available.

For complete working code, see [Pre and post-chat forms for Watson Assistant web chat](#) example.

## Tutorial: Customizing text elements based on the current page

This tutorial shows how you can dynamically customize the launcher or home screen text based on the page the user is currently viewing.

-  **Note:** For a complete, working version of the example described in this tutorial, see [Change launcher and home screen text for Watson Assistant web chat](#).

The launcher text and home screen greet the user and suggest what the user might want to do. Rather than showing the same text all the time, you might want to customize this text based on the page on your website the user is currently viewing. For example, instead of a generic greeting, a customer viewing a page about credit cards might see text that is specific to credit card services.

- Tip:** Although this example shows how to adapt the text based on the current page, you can use the same basic approach to adapt the text based on any client condition (such as the time of day or the user's geographical location).

To change the launcher or home screen text elements based on the current page the user is viewing, follow these steps:

1. Determine what page the user is currently viewing. Depending on the design of your application, there are various ways to do this, but one simple mechanism is to check the value of a URL query parameter (in this example, `page`):

```
$ const page = new URLSearchParams(window.location.search).get('page');
```

2. When the web chat is loaded, check this value and conditionally set the text based on which page is being viewed. You can use any condition that makes sense for your application. In this example, we are simply checking the value of the `page` query parameter:

```
if (page === 'cards') {  
  // Update the launcher and home screen with text that is  
  // specific to credit cards.  
  ...  
} else if (page === 'account') {  
  // Update the launcher and home screen with text that is  
  // specific to the user's account.  
  ...  
}
```

3. Change the launcher text using the [updateLauncherGreetingMessage\(\)](#) instance method. This changes the text that is displayed on the launcher (by default after 15 seconds).

```
$ instance.updateLauncherGreetingMessage("I see you're interested in credit      cards! Let me know  
if I can help.");
```

4. Change the home screen configuration using the [updateHomeScreenConfig\(\)](#) instance method:

- Make sure the home screen is enabled.

- Change the greeting based on the page the user is viewing.
- Configure the buttons shown by the home screen so they offer options that are appropriate for the page the user is viewing.

```
$ instance.updateHomeScreenConfig({  
  is_on: true,  
  greeting: 'What can I tell you about credit cards?',  
  starters: {  
    is_on: true,  
    buttons: [  
      { label: 'Card interest rates' },  
      { label: 'Cards with rewards' },  
      { label: 'Business cards' }  
    ]  
  }  
});
```

For complete working code, see the [Change launcher and home screen text for Watson Assistant web chatexample](#).

# Deploying for phone

## Integrating with phone Plus

---

IBM Cloud

Adding the phone integration to your assistant makes your assistant available to customers over the phone.

If an end user asks to speak to a person, the phone integration can transfer the call to an agent. Supported live agent and contact center integrations:

- Genesys
- Twilio Flex
- NICE CXone
- Bring your own

There are several ways to add the phone integration to your assistant:

- You can generate a free phone number that is automatically provisioned from IntelePeer. This is available only with new phone integrations. If you have an existing phone integration, you must delete it and create a new one to switch to a free phone number.
- You can connect to a contact center with live agents. For more information about setting up the integration, see [Integrating with phone and NICE CXone contact center](#).
- You can use and connect an existing number by configuring a Session Initiation Protocol (SIP) trunk from a provider such as Genesys, IntelePeer, or Twilio.

A SIP trunk is equivalent to an analog telephone line, except it uses Voice over Internet Protocol (VoIP) to transmit voice data and can support multiple concurrent calls. The trunk can connect to the public switched telephone network (PSTN) or your company's on-premises private branch exchange (PBX).

When a customer makes a phone call using the telephone number connected to your assistant, the phone integration makes it possible for your assistant to answer. The integration converts output from your assistant into voice audio by using the IBM Watson® Text to Speech service, and the audio is sent to the telephone network through the SIP trunk. When the customer replies, the voice input is converted into text by using the IBM Watson® Speech to Text service.

This feature is available only to Plus or Enterprise plan users.

**Tip:** Depending on the architecture of your existing telephony infrastructure, there are multiple ways you might integrate it with Watson Assistant. For more information about common integration patterns, read the blog post [Hey Watson, can I have your number?](#) on Medium.

## Set up the integration

You must have Manager role access to the instance and Viewer role access to the resource group to complete setup. For more information about access levels, see [Managing access](#).

To set up the integration:

1. In the **Integrations** section on the main page for your assistant under **Essential Channels**, you will see a tile for **Phone**.
2. On the **Phone** tile, click **Add**.
3. On the pop-up window, click **Add** again.
4. Choose whether to generate a free phone number for your assistant, integrate with your contact center, or connect to an

existing SIP trunk:

- To generate a free phone number for your assistant, click **Generate a free phone number**.

 **Note:** Generating a free phone number is supported only for Watson Assistant instances in the Dallas and Washington DC data centers.

- To integrate with a [contact center](#), click **Integrate with your contact center**.
- To use an existing phone number you have already configured with a [SIP trunk provider](#), click **Use an existing phone number with an external provider**.

Click **Next**.

5. If you are integrating with a contact center, follow the instructions to configure the contact center. Skip this step if you are generating a free phone number.

1. On the **Select contact center** page, select the tile of the connect center you would like to use.
2. On the **Connect to contact center** page, enter the required information. There is a **Test Connection** button on the page to validate the connection. Click **Next**.
6. If you are using an existing phone number, follow the instructions to configure the SIP trunk. Skip this step if you are generating a free phone number.
  1. On the **Bring your own SIP trunk** page, copy the SIP URI and assign it to your SIP trunk. Click **Next**.
  7. On the **Phone number** page (only for **Integrate with your contact center** and **Use an existing phone number with an external provider**), specify the phone number of the SIP trunk. Specify the number by using the international phone number format: `+1 958 555 0123`. Do not surround the area code with parentheses.

Currently, only one primary phone number can be added during initial setup of the phone integration. You can add more phone numbers in the phone integration settings later.

Click **Next**.

8. On the **Speech to Text** page, select the instance of the Speech to Text service you want to use for the phone integration.
  - If you have existing Speech to Text instances, select the instance you want to use from the list.
  - If you do not have any existing Speech to Text instances, click **Create new instance** to create a new Plus instance.
9. In the **Choose your Speech to Text language model** field, select the language model you want to use.

The list of language models is automatically filtered to use the same language as your assistant. To see all language models, toggle the **Filter models based on assistant language** switch to **Off**.

 **Note:** If you created specialized custom models that you want your assistant to use, choose the Speech to Text service instance that hosts the custom models now, and you can configure your assistant to use them later. The Speech to Text service instance must be hosted in the same location as your Watson Assistant service instance. For more information, see [Using a custom language model](#).

For more information about language models, see [Languages and models](#) in the Speech to Text documentation.

Click **Next**.

10. On the **Text to Speech** page, select the instance of the Text to Speech service you want to use for the phone integration.

- If you have existing Text to Speech instances, select the instance you want to use from the list.
- If you do not have any existing Text to Speech instances, click **Create new instance** to create a new Standard instance.

11. In the **Choose your Text to Speech voice** field, select the voice you want to use.

The list of voices is automatically filtered to use the same language as your assistant. To see all voices, toggle the **Filter voices based on assistant language** switch to **Off**.

For more information about voice options, and to listen to audio samples, see [Languages and voices](#) in the Text to Speech documentation.

Click **Next**.

**⚠ Important:** Any speech service charges incurred by the phone integration are billed with the Watson Assistant service plan as *voice add-on* charges. After the instances are created, you can access them directly from the IBM Cloud dashboard. Any use of speech instances that occurs outside of your assistant is charged separately as speech service usage costs.

The phone integration setup is now complete. On the **Phone** page, you can click the tabs to view or edit the phone integration.

**💡 Note:** If you chose to generate a free telephone number, your new number is displayed on the **Phone number** tab immediately. However, provisioning the new number might take several minutes.

## Adding more phone numbers

If you are using existing phone numbers you configured via a SIP trunk provider, you can add multiple numbers to the same phone integration.

**💡 Note:** If you generated a free phone number, you cannot add more numbers.

To add more phone numbers:

1. In the phone integration settings, go to the **Phone number** tab.

2. Use one of the following methods:

- To add phone numbers one by one, click **Add phone number** in the table, and enter the phone number along with an optional description. Click the **Add** button to save the number.
- To import a set of phone numbers that are stored in a comma-separated values (CSV) file, click the *Upload a CSV file* icon (![Add phone number][images/phone-integ-import-number.png]), and then find the CSV file that contains the list of phone numbers.

The phone numbers you upload will replace any existing numbers in the table.

## Setting up live agent escalation

If you want your assistant to be able to transfer a conversation to a live agent, you can connect your phone integration to a

contact center. For more information, see instructions for the supported platform:

- [Genesys](#)
- [Twilio Flex](#)

## Phone integration limits

Any speech service charges incurred by the phone integration are included as *Voice add-on* charges in your Watson Assistant service plan usage. The *Voice add-on* use is charged separately and in addition to your service plan charges.

Plan usage is measured based on the number of monthly active users, where a user is identified by the caller's unique phone number. An MD5 hash is applied to the phone number and the 128-bit hash value is used for billing purposes.

The number of concurrent calls that your assistant can participate in at one time depends on your plan type.

Plan	Concurrent calls
Enterprise	1,000
Plus	100
Trial	5

Plan details

## Phone integration configuration

IBM Cloud

After you have set up the phone integration for your assistant, you can modify the phone integration settings to customize the call behavior.

### Handling call and transfer failures

You can configure the phone integration to transfer the caller to a human agent if the phone connection fails for any reason. To transfer the caller to a human agent automatically, go to the **Advanced** tab in the phone integration settings, and make the following configuration selections:

- **SIP target when a call fails:** Add the SIP endpoint for your support agent service. Specify a SIP or telephone URI for a general call queue that can redirect requests to other queues. For more information, see [Configuring backup call center support](#).
- **Call failure message:** Add the message you want the assistant to say to a caller before it transfers the call to a human agent.

If, after you transfer the call to a human, the connection to a live agent fails for any reason, you can configure what to do.

- **Transfer failure message:** Add the message you want the assistant to say to a caller if transfer to a live agent fails. The message can be up to 150 characters in length.
- **Disconnect call on transfer failure:** Choose whether to disconnect the call after the failure message. This option is enabled by default. If this option is disabled, when a call transfer fails, your assistant can disconnect or process a different action.

If you choose to leave a call connected despite a transfer failure, Watson Assistant will initiate a new turn to determine the

next step. It's important that the Assistant be configured with an Action or webhook that can handle this scenario.

**Note:** The phone integration supports disaster recovery by providing the ability to do a fast failover to another region instead of routing the call to a live agent when a service outage occurs. This is accomplished by sending a SIP 503 response to the upstream SIP trunking provider, instead of auto referring the call to a live agent when failures happen during the setup of a call. This 503 response can then be used by the SIP trunking provider to reroute the call to another region. If you want to take advantage of this capability, open a service ticket against the Watson Assistant service instance that requires disaster recovery.

## Secure the phone connection

You can add security to the phone connection by going to the **Advanced options** tab in the phone integration settings, and selecting one or both of the following options:

- **Force secure trunking:** Select this option to use Secure Real-Time Transfer Protocol (SRTP) to secure the audio that is transmitted over the phone. For more information about RTP, see [Call routing details](#).
- **Enable SIP authentication:** Select this option if you want to require SIP digest authentication.

When SIP authentication is required, all inbound traffic (meaning requests from the SIP provider to your assistant) is authenticated using SIP digest authentication, and must be sent using Transport Layer Security (TLS). If this option is selected, the SIP digest user name and password must be configured, and the SIP trunk being used to connect to Assistant must be configured to use only TLS.

**Important:** If you use Twilio as your SIP trunk provider, you cannot enable SIP authentication for outbound SIP trunks to Watson Assistant.

## Applying advanced SIP trunk configuration settings

To configure how your assistant interacts with a SIP trunk from an external provider, go to the **SIP trunk** tab in the phone integration settings, and update the following options in the **SIP trunking integration** section:

- **SIP INVITE headers to extract:** List the headers that you want your assistant to use.

The SIP INVITE request can include metadata about the call in headers that can be extracted and sent to your assistant using context variables. For example, many companies use Interactive Voice Response (IVR) systems that pass information about an incoming call by using SIP headers. If you want to make use of any of these headers, list the header names here.

The specified headers, if present in the request, are stored in the context variable `sip_custom_invite_headers`, along with other related metadata that is automatically extracted from the SIP INVITE. This variable is an array in which each key/value pair represents a header from the request, as in this example:

```
$ {  
  "input": {  
    "text": "",  
    ...  
  },  
  "context" : {  
    "global" : {...},  
    "skills" : {...},  
    "integrations" : {  
      "voice_telephony": {  
        "private": {  
          ...  
        }  
      }  
    }  
  }  
}
```

```
        "user_phone_number": "+18594213456",
    },
    "sip_call_id": "Aob2-2743-5678-1234",
    "assistant_phone_number": "+18882346789",
    "sip_custom_invite_headers": {
        "X-customer-name": "my_name",
        "X-account-number": "12345"
    }
}
}
```

You can then reference these headers in your assistant. For example, you might check the header value in a step condition to determine the next step. You can also use these headers when searching the assistant logs; for example, you might search for a custom header to find all the messages associated with particular account.

- **Disable the ring that callers will hear while the assistant is contacted:** Choose whether you want the caller to hear a signal that indicates that the assistant is being contacted.

A `180 Ringing` response is sent from the assistant back to the SIP trunk provider while your assistant processes the incoming call invitation. The ringing response is sent by default.

- **Don't place callers on hold while transferring to a live agent** Choose whether the phone integration puts the caller on hold.

If your SIP trunk provider manages holds, disable this feature. For example, some SIP trunk providers prefer to have the assistant send a SIP REFER request, so they can put the call on hold themselves.

For more information about the SIP protocol, see [RFC 3261](#) and about the RTP protocol, see [RFC 3550](#).

## Configuring backup call center support

When you use the phone integration as the first line of assistance for customers, it's a good idea to have a live agent backup available. You can design your assistant to transfer a call to a human in case the phone connection fails, or if a user asks to speak to someone.

Your company might already have one or more phone numbers that connect to an automatic call dispatcher (ACD) that can queue callers until an appropriate agent is available. If not, choose a call center service to use as your backup.

A conversation cannot be transferred from one integration type to another. For example, if you use the web chat integration with service desk support, you cannot transfer a phone call to the service desk that is set up for the web chat.

You must provide the call center SIP URI for the call center service you use. You must specify this information in your assistant when you enable a call transfer from a dialog node or action step. For more information, see [Transferring a call to a live agent](#).

## Optimize your actions for phone interaction

For the best customer experience, design your dialog with the capabilities of the phone integration in mind:

- Do not include HTML elements in your action responses. To add formatting, use Markdown. For more information, see [Formatting responses](#).
- You can use a search extension to include search results in actions that the phone integration will read. When search results are returned, the phone integration reads the introductory message (for example, `I found this information that might be helpful`), and then the body of only the first search result.

The entire search response (meaning the introductory message plus the body of the first search result) must be less than 5,000 characters long or the response will not be read at all. Be sure to test the search results that are returned and curate the data collection that you use as necessary.

For more information about using the search integration, see [Leveraging existing help content](#).

For more information about how to implement common actions from your dialog, see [Handling phone interactions](#).

## Setting up a SIP trunk

If you do not use the option to generate a free phone number, you are responsible for setting up the SIP trunk that will be used by the phone integration. Find a provider and create a SIP trunk account. Your account will be charged for the phone integration's use of the SIP trunk.

You can set up a SIP trunk in the following ways:

- [Creating a Twilio SIP trunk](#)
- [Use other third-party providers](#)
- [Bring your own SIP trunk](#)
- [Migrate from Voice Agent with Watson](#)

### Creating a Twilio SIP trunk

To set up a Twilio SIP trunk, complete the following steps:

1. Create a Twilio account on the [Twilio website](#).
2. From the Twilio website, go to the *Elastic SIP Trunking* dashboard. If you do not see it on the sidebar, go to the Search Bar at the top and search for 'Elastic SIP Trunking', then select **Elastic SIP Trunks**.
3. On the **Elastic SIP Trunks** page, click the *Create new SIP Trunk* button to create a SIP trunk. Enter a name for your SIP trunk and click *Create*. If you already have a SIP trunk, go to the next step.
4. From the **Elastic SIP Trunks** page, select your SIP trunk.
5. Select *Origination* from the navigation bar for your SIP trunk and configure the origination SIP URI.

You can get the SIP URI for your phone integration from the phone integration configuration page.

6. If you plan to support call transfers, enable Call Transfer (SIP REFER) in your SIP trunk. If you expect to transfer calls to the public switched telephone network (PSTN), also enable PSTN Transfer on your trunk.
7. Select *Numbers* from the navigation bar for your SIP trunk, and then do one of the following things:
  - Click *Add a number* and then *Buy a Number*.
  - If you already have a number, you can click *Add a number* and then *Add an Existing Number*.

If you use a Lite or Trial Twilio account for testing purposes, then be sure to verify the transfer target. For more information, see the [Twilio documentation](#).

You cannot enable SIP authentication if you choose Twilio as your SIP trunk provider. Twilio doesn't support SIPS for originating calls.

## Using other third-party providers

You can ask for help setting up an account with another SIP trunk provider by opening a support request.

IBM has established relationships with the following SIP trunk providers:

- [Five9](#)
- [Genesys](#)
- [Vonage](#)
- [Voximplant](#)

The SIP trunk provider sets up a SIP trunk for your voice traffic, and manages access from allowed IP addresses. Most of the major SIP trunk providers have existing relationships with IBM. Therefore, the network configuration that is required to support the SIP trunk connection typically can be handled for you with minimal effort.

1. Create a [IBM Cloud case](#).
2. Click **Customer success** as the case type.
3. For **Subject**, enter `SIP trunk provider setup for Watson Assistant`.
4. Include the following information in the description:
  - Company Name
  - Your IBM Cloud account ID
  - Your Watson Assistant service name
  - Network diagram with IP address or SIP trunk provider information

## Bring your own SIP trunk

If you choose to use a SIP trunk carrier that IBM does not have an established relationship with, you can do so.

The following table lists the fully qualified domain names and IP addresses that are used for SIP connections.

Location	Domain names	IP addresses
Dallas	public.0001.voip.us-south.assistant.watson.cloud.ibm.com public.0002.voip.us-south.assistant.watson.cloud.ibm.com public.0003.voip.us-south.assistant.watson.cloud.ibm.com	67.228.108.82 169.63.5.162 150.239.30.146
Frankfurt	public.0001.voip.eu-de.assistant.watson.cloud.ibm.com public.0002.voip.eu-de.assistant.watson.cloud.ibm.com public.0003.voip.eu-de.assistant.watson.cloud.ibm.com	161.156.178.162 169.50.56.146 149.81.86.82
London	public.0001.voip.eu-gb.assistant.watson.cloud.ibm.com public.0002.voip.eu-gb.assistant.watson.cloud.ibm.com public.0003.voip.eu-gb.assistant.watson.cloud.ibm.com	158.176.120.162 141.125.102.34 158.175.99.34
Seoul	public.0001.voip.kr-seo.assistant.watson.cloud.ibm.com	
Sydney	public.0001.voip.au-syd.assistant.watson.cloud.ibm.com public.0002.voip.au-syd.assistant.watson.cloud.ibm.com public.0003.voip.au-syd.assistant.watson.cloud.ibm.com	168.1.47.2 135.90.86.50 168.1.106.130
Tokyo	public.0001.voip.jp-tok.assistant.watson.cloud.ibm.com public.0002.voip.jp-tok.assistant.watson.cloud.ibm.com public.0003.voip.jp-tok.assistant.watson.cloud.ibm.com	165.192.69.82 128.168.105.178 161.202.149.162
Washington, DC	public.0001.voip.us-east.assistant.watson.cloud.ibm.com public.0002.voip.us-east.assistant.watson.cloud.ibm.com public.0003.voip.us-east.assistant.watson.cloud.ibm.com	52.116.100.158 169.61.70.162 169.59.136.194

### SIP network information

## Migrating from Voice Agent with Watson

If you created an IBM® Voice Agent with Watson™ service instance in IBM Cloud to enable customers to connect to an assistant over the phone, consider using the phone integration instead. You can use the same SIP account and phone number that you configured for use with Voice Agent with Watson in the phone integration.

The phone integration provides a more seamless integration with your assistant. However, the integration currently does not support the following functions:

- Outbound calling
- Configuring backup locations
- Event forwarding to save call detail reports in the IBM Cloudant for IBM Cloud database service
- Reviewing the usage summary page. Use IBM Log Analysis instead. For more information, see [Viewing logs](#).

To migrate from Voice Agent with Watson to the Watson Assistant phone integration, complete the following steps:

1. From the Voice Agent with Watson page, copy the phone number or numbers that you used for your SIP account.
2. When you set up the Watson Assistant phone integration, add the phone number or set of numbers that you copied in the previous step.
3. From the phone integration setup page, copy the SIP uniform resource identifier (URI).
4. In your SIP trunk account, replace the Voice Agent with Watson URI that you specified previously with the URI that you copied from the phone integration setup page in the previous step.

For example, if you use a Twilio SIP trunk, you would add the assistant's SIP uniform resource identifier (URI) to the Twilio **Origination SIP URI** field.

5. If your SIP trunk provider is not already allowlisted with the Watson Assistant region you are migrating to, follow these [instructions](#) to get access to your SIP trunk.

## Call routing details

Incoming calls to your assistant follow this path:

1. A customer calls the customer support phone number that is managed by your Session Initiation Protocol (SIP) trunk provider.
  2. The SIP trunk service sends a SIP `INVITE` HTTP request to your assistant's phone integration to establish a connection.
  3. The phone integration connects to the speech services that are required to support the interaction.
  4. After the services are ready, the connection is established, and audio is sent over the Real-time Transport Protocol (RTP).
- RTP is a network protocol for delivering audio and video over IP networks.
5. The greeting action of the assistant is processed. The response text is sent to the Text to Speech service to be converted to audio and the audio is sent to the caller.
  6. When the customer says something, the audio is converted to text by the Speech to Text service and is sent to your assistant for evaluation.
  7. The assistant processes the input and calculates the best response. The response text from the assistant is sent to the Text to Speech service to be converted to audio and the audio is sent back to the caller over the existing connection.
  8. If the caller asks to speak to a person, the assistant can transfer the person to a call center. A SIP `REFER` request is sent

to the SIP trunk provider so it can transfer the call to the call center SIP URI that is specified in the dialog node where the transfer action is configured.

9. When one of the participants of the call hangs up, a SIP **BYE** HTTP request is sent to the other participant.

## Integrating with phone and Genesys Cloud

---

IBM Cloud

You can use the phone integration to help your customers over the phone and transfer them to live agents inside of Genesys Cloud. If, in the course of a conversation with your assistant, a customer asks to speak to a person, you can transfer the conversation directly to a Genesys Cloud agent.

### Before you begin

To use this integration pattern, make sure you have the following:

- Watson Assistant Plus or Enterprise Plan (required for phone integration)
- A working assistant you are ready to deploy
- A Genesys Cloud account

## Integrating with Genesys Cloud

To integrate your assistant with Genesys Cloud, follow these steps:

1. Log into the [Genesys Cloud console](#).
2. Click **Admin**.
3. On the **Telephony** tab, click **Trunks**.
4. In the **External Trunks** section, click **Create new**. Specify the following information:
  - In the **External Trunk Name** field, type a descriptive name (for example, **Watson**).
  - In the **Type** field, select **BYOC Carrier** and then **Generic BYOC Carrier**.
  - In the **Inbound SIP Termination Identifier** field, specify any name you want to use (for example, **Watson**). This value will not be used for now, but it is required by Genesys Cloud.



Topology

Metrics

**Trunks**

Sites

Edge Groups

Edges

Phone Management

Certificate Authorities

DID Numbers

Extensions

**External Trunk Name**

WatsonDoc

**Type**

BYOC Carrier

Generic BYOC Carrier

**Managed By** **Everyone**

Provider Only

**Trunk State** **In Service****Inbound / Termination****Inbound SIP Termination Identifier**

WatsonDoc

**DNIS Replacement Routing** 

Disabled

5. Under **Outbound**, scroll to the **SIP Servers or Proxies** section. Specify the following information:

- In the **Hostname or IP Address** field, type the SIP URI (not including `sips:`) from your Watson Assistant phone integration settings.
- In the **Port** field, type `5060`.

Click the `+` button.



**Note:** Currently, SIPS and digest authentication are supported.



Topology

Metrics

**Trunks**

Sites

Edge Groups

Edges

Phone Management

Certificate Authorities

DID Numbers

Extensions

**External Trunk Name**

WatsonDoc

**Type**

BYOC Carrier

Generic BYOC Carrier

**Managed By** **Everyone**

Provider Only

**Trunk State** **In Service****Inbound / Termination****Inbound SIP Termination Identifier**

WatsonDoc

**DNIS Replacement Routing** **Disabled**

6. Under **SIP Access Control**, add the [IP addresses](#) for the data center where your assistant is located.

7. Under **Identity**, specify the following information:

- Toggle the **Address Omit + Prefix** switch to **Disabled**.

**Calling**

**Address Transformation**

Match Regular Expression      Format Regular Expression

No Transformations

Match Regular Expression      Format Regular Expression +

Address Digits Length ?      Address Omit + Prefix ?  Disabled

0

8. Under **Media**, remove **Opus** from the **Preferred Codec List**. Click **Select a Codec** and then select **g729** to add it to the list. Leave **PCMU** as the first item in the list.

▼ **Media**

**DSCP Value** ?      **Media Method** ?

2E (46, 101110) EF      Normal

⚠ G.729 is not recommended over WAN links that don't guarantee QoS. (ex: the Internet)

**Preferred Codec List** ?      **SRTP Cipher Suite List** ?

↑ ↓ audio/PCMU	↑ ↓ AES_CM_128_HMAC_SHA1_80
↑ ↓ audio/PCMA	
↑ ↓ audio/g729	

Select a Codec      Select a Cipher Suite

9. Under **Protocol**, enable **Take Back and Transfer**.

10. Click **Save External Trunk**.

11. Under **Sites**, select the existing site you want to use this trunk with. To create a new site, specify a name and location and click **Create**.

12. Click **Number plans**. Create a new number plan and specify the following information:

- In the **Number Plan Name** field, type a descriptive name (for example, **Watson**).
- For **Match type**, select **E. 164 Number List**.
- In the **Numbers** field, type a number in the **Start** and **End** fields. This does not need to be a real number; you can make up any number to use as an identifier to assign to Watson. Specify the same number in both fields.

**Note:** To create a PSTN number you can give to your clients, you must create a Direct Inward Dialing (DID) or Bring Your Own Carrier (BYOC) number. For more information about how to do this, see the Genesys documentation.

- 1. In the **Classification** field, type a classification name (for example, **Watson**).

Click **Save Number Plans**.

Number Plan Name: WatsonDoc

Match Type: E.164 Number List

Numbers: +1 408-981-3165 → +1 408-981-3165 X

Classification: Watson

Start Number → End Number +

Save Number Plans Cancel

13. Click **Outbound Routes**. You can either edit the default outbound route or create a new one. Specify the following information:

- In the **External Trunks** field, click **Select External Trunks**. Select the trunk you created for Watson Assistant.
- In the **Classifications** field, add the applicable classifications. This should at least include **National** and the classification you created for Watson Assistant earlier. (The **National** route is used only to simulate the call in order to make sure the trunk is operational.)
- Toggle the **State** switch to **Enabled**.

## + New Outbound Route

Default Outbound Route

Outbound Route Name

Default Outbound Route

Description

State

Enabled

Classifications

Emergency x

National x

International x

Network x

Watson x

Save Outbound Routes

Cancel

14. Click **Save Outbound Routes**.

15. Go to the **Simulate Call** tab and click the **Simulate Call** button. The trunk should be shown as operational. No actual call is made during simulation.

- i** Simulate call will use settings from the "General", "Number Plans", and "Outbound Routes" tabs. You do not have to test before applying the changes.



**✓ Success**

**Normalized Number** [?](#)

✓ tel:+14089813165

**Number Plan** [?](#)

✓ WatsonDoc

**Classification** [?](#)

✓ Watson

**Outbound Route** [?](#)

✓ Default Outbound Route

**External Trunks** [?](#)

WatsonDoc

✓ This Trunk is operational on all of the associated sites.

**Preferred Edges** [?](#)

None

**Additional Edges**

- virtual-edge-i-0e13cd366bb0dd58f - Port 1
- virtual-edge-i-00a82f1136a30435c - Port 1

---

16. Go to **Phone Management** and click **Create new**. Specify the following information:

- In the **Phone Name** field, specify a descriptive name.
- In the **Base Settings** field, select **WebRTCPhone**.
- In the **Site** field, select the site you want to use.
- In the **Person** field, select yourself.

17. In the Watson Assistant user interface, [create a new phone integration](#). Specify the following information:

- When prompted, select **Use an existing phone number with an external provider**.
- Specify the phone number you assigned in the Genesys **Number Plans** setting. This is not necessarily a real phone number; it is just the identifier you assigned.
- Complete the phone integration setup process. For more information, see [Integrating with phone](#).
- After the phone integration is set up, go to the **SIP trunk** tab and uncheck the **Don't place callers on hold while transferring to a live agent** option.

18. In the Genesys Cloud console, click the circle in the upper left corner. Select **Phone**, and then choose the phone you

created in the **Phone management** section. Set yourself as available. The phone icon on the left should now be active.

19. Click **+** to start a new call. Specify the number you assigned to Watson Assistant and then click **Dial**. You should now hear your assistant speak.



**Note:** If you encounter any errors, click **Performance -> Interactions** and view the PCAP file to read the diagnostics.

## Transferring to a live agent

Now that your Genesys Cloud environment can connect to Watson Assistant, you can set up the ability for your assistant to transfer calls back to your live agents. To do so, follow these steps:

1. In the Genesys Cloud console, go to **DID Numbers -> DID Ranges** and create a new range. Specify the following information:
  - In the **DID Start** and **DID End** fields, specify a phone number. (Once again, you do not need to use a real phone number; you can just make up an identifier for your Genesys environment, such as **+1-888-888-1234**.)

Provider	Comments	

Create Range

DID Start

+1 +18888881234

DID End

+1 +18888881234

Service Provider

Watson

Comments

1 – 1 of 1 DID Ranges

Save Cancel

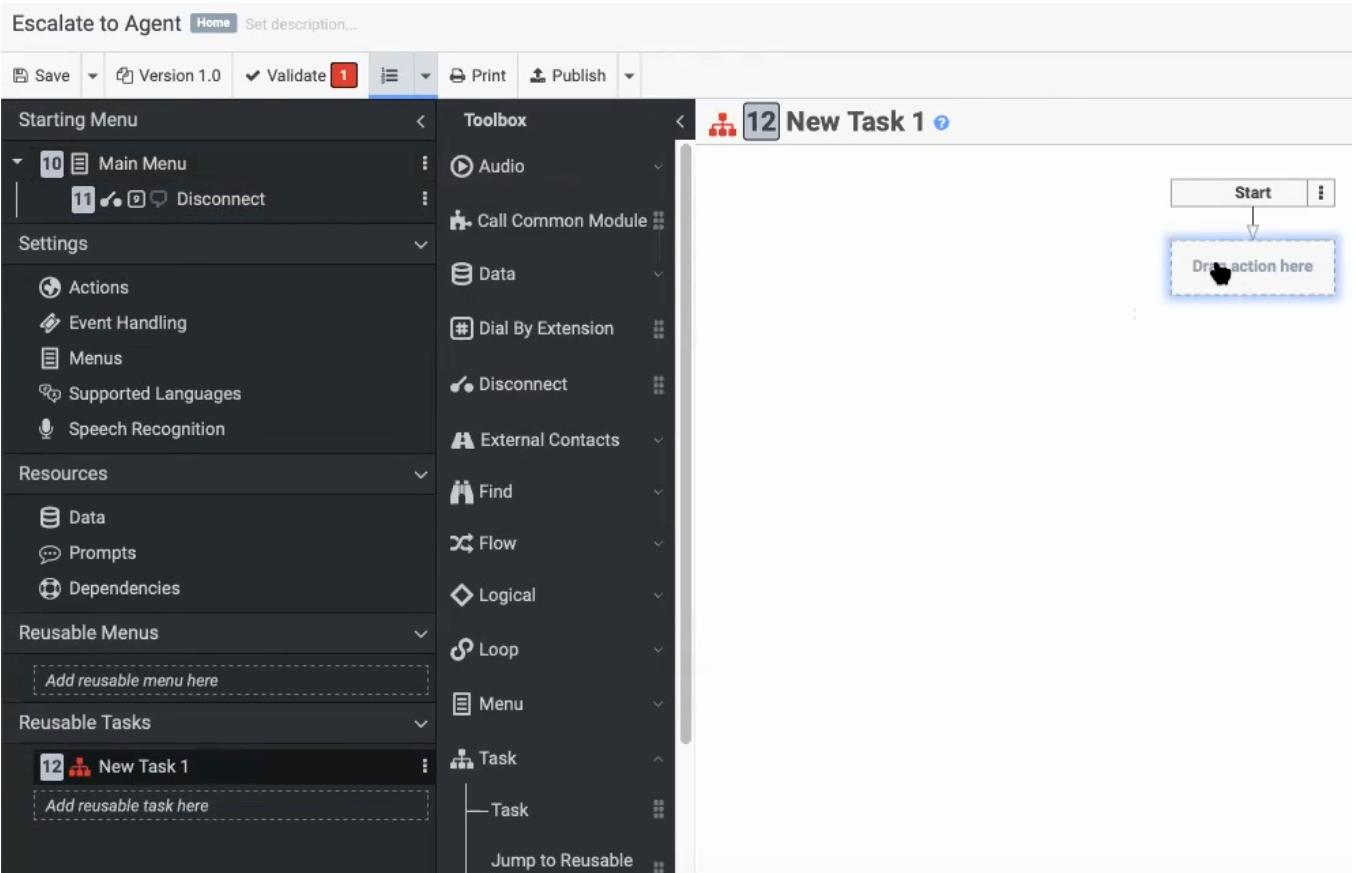
- In the **Service Provider** field, type a descriptive name (for example, **Watson**).

2. If you have not already set up a queue to enable callers to wait for available agents, follow these steps to create a simple one now:

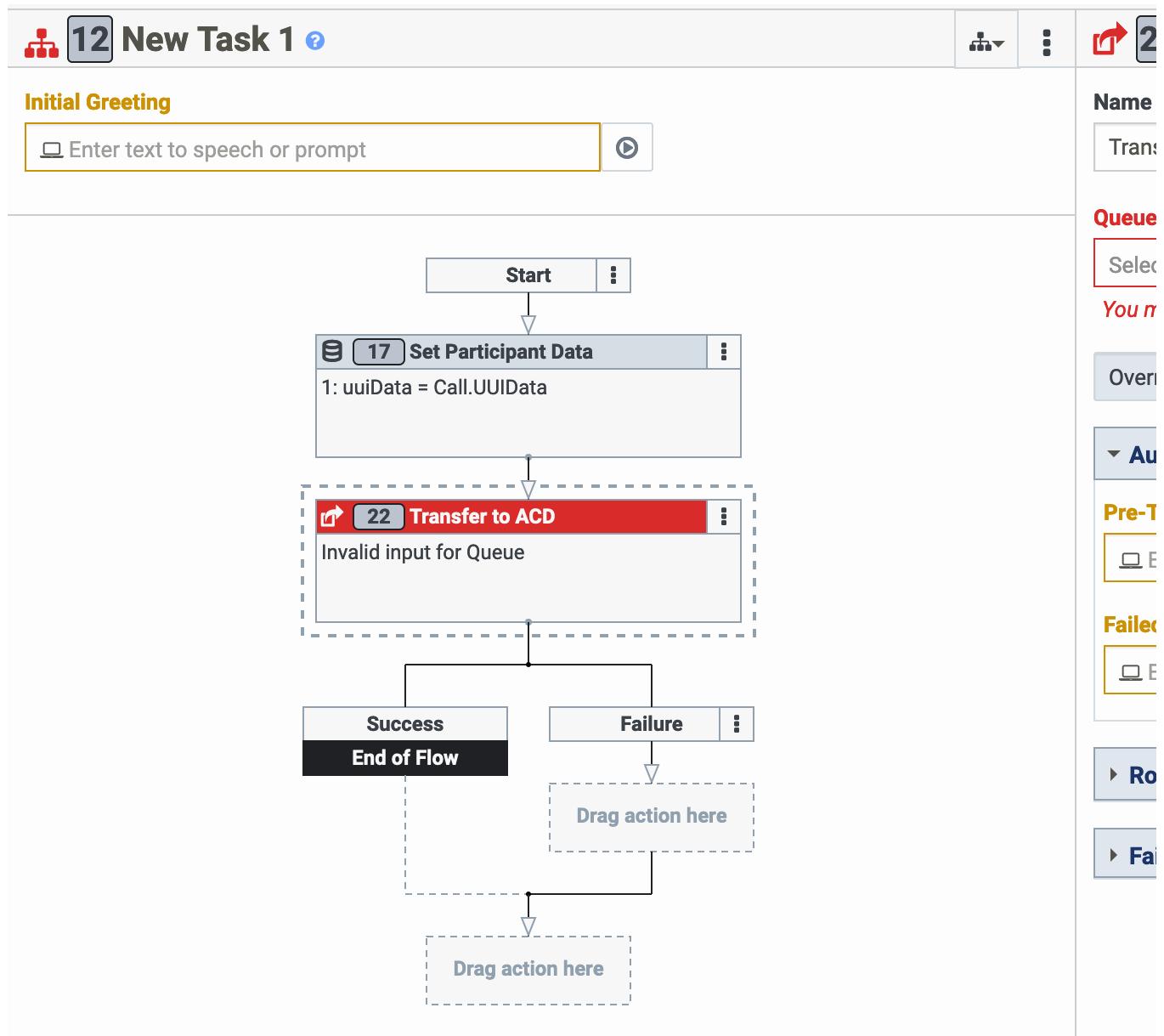
1. Click **Admin**.
  2. Under **Contact Center**, click **Queues**.
  3. Create a new queue and give it a descriptive name.
  4. Add yourself as a member.
  5. Click **Save**.
3. Create a simple call flow. Your business might already have something more complex for routing.
1. Click **Admin**.
  2. Click **Architect**.
  3. In the **Flows: Inbound Call** section, click **+** to create a new flow. Give it a descriptive name (for example, **Escalate to Agent**).

The screenshot shows the 'Escalate to Agent' call flow configuration. The left sidebar contains sections for 'Starting Menu', 'Settings' (Actions, Event Handling, Menus, Supported Languages, Speech Recognition), 'Resources' (Data, Prompts, Dependencies), and 'Reusable Menus' (with a placeholder 'Add reusable menu here'). The main workspace is titled '10 Main Menu' and includes sections for 'Initial Greeting' (Hello, this is the initial greeting), 'Menu Prompt' (You are at the Main Menu, press 9 to disconnect), 'Default Menu Choice' (None (disconnect the interaction)), 'Menu Options', and 'Speech Recognition Options'. The 'Toolbox' on the right lists available actions: Dial By Extension, Disconnect, Menu, Task, and Transfer.

1. In the toolbox, click **Task** and drag it into **Reusable Tasks**.



1. From your toolbox, under **Transfer**, drag the **Transfer to ACD** widget into the first action.



1. Select the queue you want to use.
2. In the toolbox, click the **Disconnect** widget and drag it into the action after **Failure**. (This disconnects the call if the transfer fails.)
3. Click the three dots under **Reusable tasks** and click **Set this as the starting task**. You can remove the **Initial Greeting**, because your assistant will speak before handing off the call.
4. Click **Publish** in the menu bar to make this transfer live.
5. Return to the main Genesys Cloud console.
6. Click **Admin** and then navigate to **Call Routing** in the **Routing** section.
7. Give the route a descriptive name (for example, `Escalate to Agent`).
8. Under **Regular Routing**, for all calls, select your new flow.

9. Assign the DID number you previously created.

10. Click **Save**.

4. Make sure your assistant is configured to transfer calls to an agent using the *Connect To Agent* response\_type. For more information about how to do this, see [Transferring a call to a live agent](#)

For the `sip.uri` parameter, use the DID number you created in Genesys Cloud, as well as the inbound SIP URI from your Genesys trunk. Use the following format:

```
$ {
  "generic": [
    {
      "response_type": "connect_to_agent",
      "transfer_info": {
        "target": {
          "service_desk": {
            "sip": {
              "uri": "sip:+18883334444\\@example.com",
              "transfer_headers_send_method": "refer_to_header"
            }
          }
        }
      },
      "agent_available": {
        "message": "Ok, I'm transferring you to an agent"
      },
      "agent_unavailable": {
        "message": ""
      }
    }
  ]
}
```

 **Note:** Make sure you use the `\\"` escape characters so Watson Assistant does not misinterpret the `@` as part of the entity shorthand syntax.

5. Make a test call and say something that initiates a transfer to an agent. In your Genesys Cloud console, you should see the transfer take place.

## Integrating with phone and NICE CXone contact center

IBM Cloud

Connect your assistant to a NICE CXone contact center with live agents.

Transfer customers from a chat with your assistant to live agents who can help them by phone. If customers ask to speak to someone, your assistant can forward them directly to customer support with the conversation history.

This integration creates a connection between your assistant and a contact center using NICE CXone.

You need a Plus or Enterprise Plan to use this feature.

### **Before you begin**

You must have a NICE CXone account and phone numbers allocated for this integration.

1. Go to the [NICE website](#).
2. Create an account.

3. Follow the instructions to get phone numbers or select existing phone numbers.

## Generate NICE CXone access keys

Access keys are used for authentication and consist of two parts: an access key ID and a secret access key.

To generate NICE CXone access keys to use with your assistant:

1. Log in to the NICE CXone console.

2. Click the app selector  and select **Admin**.

3. Click **Users**, and then locate and click the user account you want to use for the integration.

4. Click the **Access Keys** tab.

5. Click **Generate New Access Key**.

6. Click **Show Secret Key**, and copy the secret key to a secure location.

You cannot retrieve the secret key again after you complete the next step and **Save**. You must generate a new key if the current one is lost or forgotten.

7. Click **Save**.

## Set up the integration

To complete setup, you must have an assistant ready to deploy, your NICE CXone access keys, and phone numbers allocated for this integration.

To integrate your assistant with NICE CXone:

1. In the **Integrations** section on the main page for your assistant under **Essential Channels**, you will see a tile for **Phone**.

2. On the **Phone** tile, click **Add**.

3. On the pop-up window, click **Add** again.

4. Select **NICE CXone** on the **Select contact center** page.

Click **Next**.

5. On the **Connect to contact center** page, specify the following values: - the **Authentication URL** from NICE CXone - the **API URL**, which is the *Admin* API endpoint from NICE CXone - the **Access key ID** - the **Access key secret**

Click **Test connection** to verify the credentials.

Click **Next**.

6. On the **Phone number** page, enter a phone number you allocated for the NICE CXone integration. You can add more phone numbers later.

Click **Next**.

7. On the **Speech to Text** page, select the instance of the Speech to Text service you want to use.

- If you have existing Speech to Text instances, select the instance you want to use from the list.

- If you do not have any existing Speech to Text instances, click **Create new instance** to create a new Plus or Enterprise instance.

8. In the **Choose your Speech to Text language model** field, select the language you want to use.

The list of language models is automatically filtered to use the same language as your assistant. To see all language models, toggle the **Filter models based on assistant language** switch to **Off**.

 **Note:** If you created specialized custom models that you want your assistant to use, choose the Speech to Text service instance that hosts the custom models now, and you can configure your assistant to use them later. The Speech to Text service instance must be hosted in the same location as your Watson Assistant service instance. For more information, see [Using a custom language model](#).

For more information about language models, see [Languages and models](#) in the Speech to Text documentation.

Click **Next**.

9. On the **Text to Speech** page, select the instance of the Text to Speech service you want to use.

- If you have existing Text to Speech instances, select the instance you want to use from the list.
- If you do not have any existing Text to Speech instances, click **Create new instance** to create a new Standard instance.

Click **Next**.

10. On the **Contact center integrations** page, click **Test connection** below the setup information you entered.

- If **Invalid**, check your credentials and enter each again.
- If the credentials are correct, the **Save and exit** button becomes clickable.

Click **Save and exit**.

The connection between your assistant and NICE CXone is complete.

## Configuring the NICE CXone script

NICE CXone provides a scripting tool that allows workflow developers to define routing flows for their contact centers in CXone.

The following actions and settings in the workflow are necessary for integration to work properly.

### Connecting a caller to your assistant

Use the [Sipputhandler](#) action. In the **headerName** property, enter the name of the SIP header field that will contain the Contact ID. This header field is included in outgoing SIP INVITE messages to Watson Assistant.

- **headerName** X-Contact-ID
- **headerValue** {ContactId}

 **Note:** [Sipputhandler](#) must be executed before [Placecall](#).

Use a [Placecall](#) action to initiate an outbound call to Watson Assistant. In the **PhoneNumber** property, enter the phone number you allocated for this integration.

**Note:** The phone number must match the number you configured in [Use an existing phone number with an external provider](#) in the Watson Assistant user interface.



## Transferring a caller to a live agent

You can configure your assistant to transfer a customer to a NICE CXone live agent.

The phone integration uses the [signal](#) REST API. The **p1** attribute is preserved for the session history key. The key can be used to fetch the conversation history and present it to a live agent.

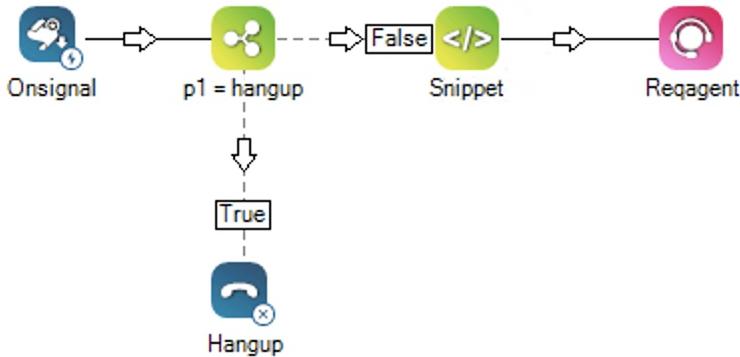
1. Use [Onsignal](#) to process the signal event.
2. Save the value of the **p1** attribute:

```
$ sessionKey = "{p1}"
```

For example, you can use the [Snippet](#) action:

```
$ ASSIGN sessionKey = "{p1}"
```

Use [Reqagent](#) to transfer a call to a live agent.



## Displaying conversation history to a live agent

Configure your script to provide a transcript of the assistant conversation to a live agent in a pop-out window. This helps the agent better understand and address a customer's needs.

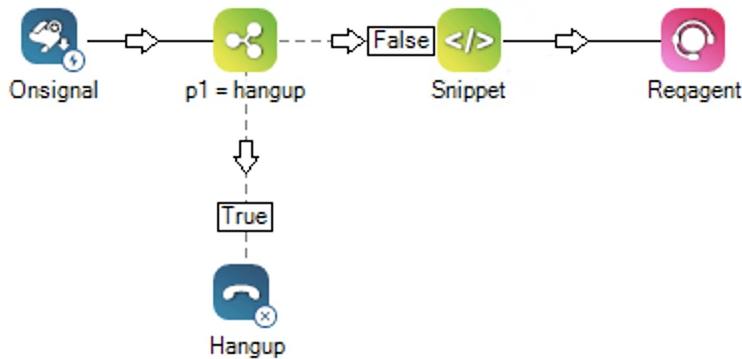
1. Add an [Onanswer](#) event to your script. The **Onanswer** event is triggered when an agent answers the call.
2. Use [Assign](#) to store the link to the conversation history into a variable. **Variable** `watson_url` **Value** `https://web-chat.global.assistant.watson.appdomain.cloud/loadAgentAppFrame.html?session_history_key={sessionKey}`
3. Use the [PopURL](#) action to display the conversation transcript to a live agent. **URL** `{watson_url}`



## Disconnecting a call

The phone integration uses the same [signal](#) REST API for disconnecting a call.

The **p1** attribute is set to `hangup`. You need to design your script so it can distinguish between transferring a call to a live agent and disconnecting a call. If **p1** is set to `hangup` when an **Onsignal** event is triggered, use [Hangup](#) to terminate a script.



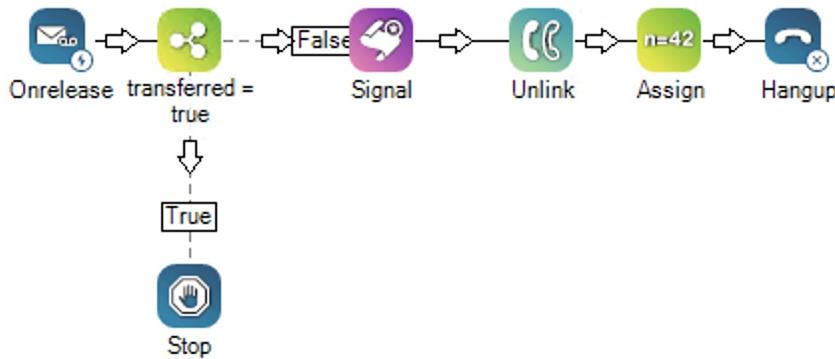
## Transferring to a live agent when an error occurs

If an error occurs during a conversation, the phone integration disconnects the call by sending a `SIP_BYE` request.

Use [Onrelease](#) to process the `BYE` request and transfer a call to a live agent.

In this example, when **Onrelease** is triggered, the script verifies whether the call was already transferred. If not, it calls the **Signal** action and sets an indication that the call is being transferred to a live agent. The indication is set using the **Assign** action.

- **Variable** transferred
- **Value** true



If the **Hangup** action is executed and an [Onrelease](#) event action is present, CXone will hang up on the caller, and the script will jump to the **OnRelease** action. Design your script so it can distinguish whether the **OnRelease** event is triggered due to a transfer or hangup.

## Adding transfer support to your assistant

Configure your assistant to transfer calls to an agent using the *Connect To Agent* response\_type. For instructions, see [Transferring a call to a live agent](#).

Use the following format:

```
{  
  "generic": [  
    {  
      "response_type": "connect_to_agent",  
      "transfer_info": {  
        "target": {  
          "nice_cxone": {  
            "custom_data": {  
              "p2": "test"  
            }  
          }  
        }  
      },  
      "agent_available": {  
        "message": "Ok, I'm transferring you to an agent."  
      },  
      "agent_unavailable": {  
        "message": "Agent is unavailable."  
      }  
    }  
  ]  
}
```

Parameters listed in the `custom_data` object are transferred to the [signal](#) REST API.

Supported parameters are `p2`, `p3`, ... `p9`. Note that `p1` is preserved and used by the phone integration for passing the session history key into the NICE CXone script.

## Integrating with phone and Twilio Flex

IBM Cloud

You can use the phone integration to help your customers over the phone and transfer them to live agents inside of Twilio Flex. If, in the course of a conversation with your assistant, a customer asks to speak to a person, you can transfer the conversation directly to a Twilio Flex agent.

### Before you begin

To use this integration pattern, make sure you have the following:

- Watson Assistant Plus or Enterprise Plan (required for phone integration)
- A Twilio account with the following products:
  - Twilio Flex
  - Twilio Voice with Programmable Voice API
  - Twilio Studio

### Adding the Watson Assistant phone integration

If you have not already added the phone integration to your assistant, follow these steps:

1. In the **Integrations** section on the main page for your assistant under **Essential Channels**, you will see a tile for **Phone**.
2. On the **Phone** tile, click **Add**.

3. On the pop-up window, click **Add** again.
4. Select **Use an existing phone number with an external provider**.
5. Complete the phone integration setup process. (For more information, see [Integrating with phone](#).)

For now, this is all you need to do. For more information about configuring the phone integration, see [Integrating with phone](#).

## Adding the Twilio Flex Project

If you don't already have a Twilio Flex project, you can create one by following these steps:

1. From the project drop-down menu, click **Create New Project**. Specify a name for the project and verify your account information.
2. On the welcome page, select Flex as the Twilio product for your new project. Fill out the questionnaire and then click **Get Started with Twilio**.

After your Flex project has been provisioned, return to the [Twilio console](#). Make sure you have selected the correct project from the drop-down list.

3. In the navigation menu, click the **All Products & Services** icon.
4. Click **Twilio Programmable Voice > Settings > General**
5. Under **Enhanced Programmable SIP Features**, toggle the switch to **Enabled**.

## Creating the call flow

After you have your phone integration and Twilio Flex project configured, you must create a call flow with Twilio Studio and provision (or port) the phone number you want your assistant to work with.

To create the call flow:

1. In the navigation menu, click the **All Products & Services** icon.
2. Click **Studio**.
3. Click **+** to create a new flow.
4. Name the new flow and then click **Next**.
5. Select **Start From Scratch** and then click **Next**.
6. At this point you should have a **Trigger** widget at the top of your flow canvas.
7. Click the **Trigger** widget.
8. Make note of the value from the **WEBHOOK URL** field. You will need this value in a subsequent step.

## Configuring the phone number

1. In the navigation menu, click the **All Products & Services** icon.
2. Click **Phone Numbers**.
3. Under **Manage Numbers**, configure the phone number you want your assistant to use. Select **Buy a Number** to buy a new

number, or **Port & Host** to port an existing phone number.

4. In the **Active Numbers** list, click the new phone number.

5. Under **Voice and Fax**, configure the following settings:

- For **CONFIGURE WITH** field, select **Webhook, TwiML Bins, Functions, Studio, or Proxy**.
- For **A CALL COMES IN**, select **Studio Flow**. Select your flow from the drop-down list.
- For **PRIMARY HANDLER FAILS**, select **Studio Flow**. Select your flow from the drop-down list.

6. Go to the Watson Assistant user interface, open the phone integration settings for your assistant.

7. In the **Phone number** field, type the phone number you configured in Flex Studio.

8. Click **Save and exit**.

## Test your phone number

You can now test that your phone number is connected to your flow by triggering a **Say/Play** widget in the Twilio Flex Flow editor.

1. Drag a **Say/Play** widget onto your flow canvas.

2. Configure the Say/Play widget with a simple phrase like `I'm alive..`

3. Connect the **Incoming call** node on your **Trigger** widget to your **Say/Play** widget.

4. Call your phone number. You should hear your Twilio flow respond with your test phrase.

5. Delete the **Say/Play** widget and continue to the next step.

6. If this test did not work as expected, double check your phone number configuration to make sure its attached to your flow.

## Creating a Twilio function to handle incoming calls

Now we need to configure the call flow to direct inbound calls to the assistant using a Twilio function. Follow these steps:

1. In the navigation menu, click the **All Products & Services** icon.

2. Click **Services**.

3. Click **Create Service**. Specify a service name and then click **Next**.

4. Click **Add > Add Function** to add a new function to your service. Name the new function `/receive-call`.

5. Replace the template in your `/receive-call` function with the following code:

```
$ exports.handler = function(context, event, callback) {  
  const VoiceResponse = require('twilio').twiml.VoiceResponse;  
  const response = new VoiceResponse();  
  const dial = response.dial({  
    answerOnBridge: "true",  
    referUrl: "/refer-handler"  
  });  
  const calledPhoneNumber = event.Called;  
  dial.sip(`sip:${calledPhoneNumber}@{sip_uri_hostname};secure=true`);  
};
```

```
    return callback(null, response);
}
```

- Replace {sip\_uri\_hostname} with the hostname portion of your assistant's phone integration SIP URI (everything that comes after sips:). Note that Twilio does not support SIPS URIs, but does support secure SIP trunking by appending ;secure=true to the SIP URI.

6. Click **Save**.

7. Click **Deploy All**.

## Redirecting to the incoming call handler

In this section you will use a TwiML **Redirect**\*\* widget in your Studio Flow editor to call out to the `/receive-call` function created in the previous section.

1. Add a **TwiML Redirect** widget to your Studio Flow canvas.
2. Connect the Incoming Call trigger to your **TwiML Redirect** widget.
3. Configure the **TwiML Redirect** widget with the URL for the `/receive-call` function you created in the previous section.
4. Your flow should now redirect to Watson Assistant when receiving an inbound call.
5. If the redirect fails, make sure you deployed your `/receive-call` function.

## Creating a Twilio function to handle transfers from assistant

We also need to configure the call flow to handle calls being transferred from the assistant back to Twilio Flex, for cases when customers ask to speak to an agent. To show this, we will use a **Say/Play** after the **TwiML Redirect** widget to show that the call is transferred back to the flow from Watson Assistant. Note that there are many things like queuing the call for a live agent that can happen at this point. These will be discussed below.

1. Add a new **Say/Play** widget to your canvas and configure it with a phrase like `Transfer from Watson complete`.
2. Connect the **Return** node on the **TwiML Redirect** widget to your **Say/Play** widget.
3. Click the **Trigger** widget.
4. Copy the value from the **WEBHOOK URL** field. You will need this value in a subsequent step.
5. On the Twilio Functions page, click **Add > Add Function** to add another new function to your service. Name this new function `/refer-handler`.
6. Replace the template in your `/refer-handler` function with the following code:

```
$ exports.handler = function(context, event, callback) {
  // This function handler will handle the SIP REFER back from the Watson Assistant Phone
  // Integration.
  // Before handing the call back to Twilio, it will extract the session history key from the
  // User-to-User header that's part of the SIP REFER Refer-To header. This session history key
  // is a string that is used to load the agent application in order to share the transcripts of
  // the caller
  // with Watson Assistant to the agent.
  // See https://github.com/watson-developer-cloud/assistant-web-chat-service-desk-
  // starter/blob/main/docs/AGENT_APP.md
  const VoiceResponse = require('twilio').twiml.VoiceResponse;

  const STUDIO_WEBHOOK_URL = '{webhook_url}';
```

```

let studioWebhookReturnUrl = `${STUDIO_WEBHOOK_URL}?FlowEvent=return`;

const response = new VoiceResponse();
console.log("ReferTransferTarget: " + event.ReferTransferTarget);

const referToSipUriHeaders = event.ReferTransferTarget.split("?")[1];
console.log(referToSipUriHeaders);
if (referToSipUriHeaders) {
  const sanitizedReferToSipUriHeaders = referToSipUriHeaders.replace(">", " ");
  console.log("Custom Headers: " + sanitizedReferToSipUriHeaders);

  const sipHeadersList = sanitizedReferToSipUriHeaders.split("&");

  const sipHeaders = {};
  for (const sipHeaderSet of sipHeadersList) {
    const [name, value] = sipHeaderSet.split('=');
    sipHeaders[name] = value;
  }
}

const USER_TO_USER_HEADER = 'User-to-User';

// Extracts the User-to-User header value
const uuidData = sipHeaders[USER_TO_USER_HEADER];

if (uuidData) {
  const decodedUUIData = decodeURIComponent(uuidData);
  const sessionHistoryKey = decodedUUIData.split(';')[0];
  // Passes the session history key back to Twilio Studio through a query parameter.
  studioWebhookReturnUrl =
`${studioWebhookReturnUrl}&SessionHistoryKey=${sessionHistoryKey}`;
}

response.redirect(
  { method: 'POST' },
  studioWebhookReturnUrl
);

// This callback is what is returned in response to this function being invoked.
// It's really important! E.g. you might respond with TWiML here for a voice or SMS response.
// Or you might return JSON data to a studio flow. Don't forget it!
return callback(null, response);
}

```

Replace `{webhook_url}` with the **WEBHOOK URL** value you copied from the **Trigger** widget in your Studio Flow.

7. Click **Save**.

8. Click **Deploy All**.

9. After you create this refer-handler, copy the function URL back into the `/receive-call` handler's **referUrl** field.

## Configuring the assistant to transfer calls to Twilio Flex

Now we need to configure the assistant to transfer calls to Twilio Flex when a customer asks to speak to an agent. Follow these steps:

1. In the Watson Assistant user interface, open the dialog skill of your assistant.
2. Add a node with the condition you want to trigger your assistant to transfer customers to an agent.
3. Add a text response to the node, and specify the text you want your assistant to say to your customers before it transfers them to an agent.

4. Open the JSON editor for the response.

5. In the JSON editor, add a [connect\\_to\\_agent](#) response, specifying your phone number as the `sip.uri` (replace `{phone_number}` with the phone number of your SIP trunk):

```
$ {
  "generic": [
    {
      "response_type": "connect_to_agent",
      "transfer_info": {
        "target": {
          "service_desk": {
            "sip": {
              "uri": "sip:{phone_number}@flex.twilio.com",
              "transfer_headers_send_method": "refer_to_header"
            }
          }
        },
        "agent_available": {
          "message": "Ok, I'm transferring you to an agent"
        },
        "agent_unavailable": {
          "message": ""
        }
      }
    ]
  }
}
```

Note that this example does not show how to use the context passed from Watson Assistant to Twilio Flex. You can reference the User-to-User information from within the Twilio Flex flow as follows:

```
$ {
  "context": {
    "widgets": {
      "redirect_1": {
        "User-to-User": "value",
      }
    }
  }
}
```

where `redirect_1` is the name of your redirect widget. For example, if you set up multiple queues, you might want to use a Twilio Split widget to pick a queue based on the returned context.

## Test your assistant

Your assistant should now be able to answer phone calls to your phone number and transfer calls back to your Twilio Flex flow. To test your assistant:

1. Call your phone number. When the assistant responds, ask for an agent.
2. At this point you should hear the phrase configured in the [Say/Play](#) widget (such as "Transfer from Watson complete").
3. If the transfer fails, use the console log to follow the flow of the call as it moves from the flow to the `/receive-call` handler, to Watson Assistant, to the refer-handler and back to your Twilio Flex flow.

## Share the conversation history with service desk agents

To enable the service desk agent to get a quick view of the conversation history between the visitor and the assistant, set up the Watson Assistant Agent App app for your Twilio Flex environment. For more information, see the documentation for the [Twilio](#)

## Troubleshooting and Logs

---

### Troubleshooting

Find solutions to problems that you might encounter while using the integration.

- If you get a *Forbidden* message: This means the phone number that you specified when you configured the phone integration cannot be verified. Make sure the number fully matches the SIP trunk phone number.
- There is a lot of latency between caller questions and Watson answers: This problem is most likely coming from latency that is caused by one of the Watson services. You can view logs in IBM Log Analysis, see [Viewing logs](#). At the end of each call, you will see a `CWSGW0160I: Call was ended.` event. Expand on this entry to see a summary of the `max_response_milliseconds` as well as details of the `assistant_interaction_summaries`. This will help you identify which service the latency is coming from.

If your plan allows, you can also look at the Call Detail Record (CDR) to determine which service is misbehaving. For more information, see [Call Detail Records \(CDRs\)](#).

### Viewing logs

The log events that occur in the components that are used by the phone integration are written to IBM Log Analysis. To check the logs, create an instance and configure the platform logs to observe the region where your service instance is hosted.

For more information about setting up an instance, see [Provisioning an instance](#).



**Note:** Currently, the Phone and SMS integrations are the only components of your assistant that write logs to the IBM Log Analysis dashboard.

After you create the instance, get log information by completing the following steps:

1. Go to the [IBM Cloud Logging](#) page.
2. Click **Options**, then choose **Edit platform**.
3. Select the region and instance, and then click **Select**.
4. To open the IBM Log Analysis console, click **Open Dashboard**.
5. The source name of the log events is *Watson*.

You can apply filters or search the logs by values such as a phone number or instance ID.

### Call Detail Records (CDRs)

The phone integration can generate call detail record (CDR) events, which contain summary information about a single call. Call detail records are configured through a webhook. For more information, see [Logging activity with a webhook](#).

For detailed information about the structure of the CDR event payload, see [CDR log event reference](#).

You can also inject custom data into the CDR event. For more information, see [Injecting custom values into CDR log events](#).

# Deploying to other channels

## Deploying your assistant

IBM Cloud

Watson Assistant routes your customer's questions and requests to the correct resolution source. However, before your assistant can properly route requests, you must complete the following steps:

1. Write content for your assistant
2. Publish the content
3. Connect an integration to your assistant
4. Deploy the assistant to a channel

After you connect an integration and deploy to a channel, you make your assistant available to your customers. You can find an overview visual of your progress toward completing these steps on the [Live environment](#) page:

The screenshot shows the Watson Assistant interface for managing environments. The top navigation bar has tabs for 'Draft' and 'Live', with 'Live' selected. The main content area is titled 'Live environment' and includes a description: 'Use the live environment for deployment to customers. It contains your published content and channel integrations where customers interact with your assistant.' Below this, there's a 'Channels' section with 'Web chat' and 'Phone' listed, and a 'Content' section showing version V4 from 03/01/2023 at 11:46AM, with a 'Live' status indicator. On the right, there's an 'Extensions' section for 'Search', which is powered by IBM Watson Discovery and allows extending what the assistant can answer by searching existing documents. A central graphic features a blue Watson logo with radiating lines.

## Reviewing your connected channels

Channels represent the locations or communication platforms where your assistant interacts with your users. Common examples of channels include the phone, a website, or Slack. If you do not connect your assistant to a channel, your users are not able to access the assistant.

You can review your connected channels in your environments, for example:

- **Draft environment:** Channels that are connected to this environment are exposed only to your internal team for testing and not to your customers.
- **Live environment:** Channels that are connected to this environment represent the public-facing experience of your assistant and are exposed to your customers.

- **Multiple environments:** If you are using [multiple environments](#), channels are exposed only to your internal team for testing and not to your customers.

For more information on environments, see [Environments](#) and [Adding and using multiple environments](#).

You can test integrations from the draft environment and interact with your draft web chat on the **Preview** page. When you first create a new assistant, the assistant automatically connects to the web chat channel in your live environment. However, the assistant itself is not available to your customers until you embed the web chat JavaScript in the header of your website. For more information, see [Adding the web chat to your website](#).

## Connecting your assistant to a new channel

All of the channels you can connect your assistant to are available in the **Integrations** catalog. Two types of integrations are available from the **Integrations** catalog:

- **Channels:** The location where your assistant interacts with your users, for example, over the phone, on a website, or in Slack. At least one channel is required for every assistant.
- **Extensions:** Add-ons to the end experience that help solve specific user problems, for example, connecting to a human agent or searching existing help content. Extensions are not required for an assistant, but they are recommended.

When you add a channel to your assistant, at least two instances of the channel are created. One instance of the channel is connected to the draft environment and the other instance is connected to the live environment. If you are using [multiple environments](#), instances of the channel are added to your extra environments. To connect your assistant to a new channel, go to the **Integrations** catalog. For more information about adding integrations to your assistant, see [Adding integrations](#).

Although a channel always exists in environments, you can configure your integration separately in each environment. For example, this allows you to test integrations on your draft environment before you go live with any integration configuration. After you add an integration, you must set it up to use it with your assistant. The **Finish setup** icon appears on any integration that you added but didn't yet set up.

You have multiple options for deploying your assistant, depending on how you want your customers to interact with it. In most cases, an assistant is deployed by using one of the following integrations:

- [Web chat integration](#): The web chat integration provides a secure and highly customizable widget you can add to your website. You can configure how and where the web chat widget appears, and you can use theming to align it with your branding and website design. If a customer needs help from a person, the web chat integration can transfer the conversation to an agent.
- [Phone integration](#): The phone integration enables your assistant to converse with customers on the phone by using the IBM Watson Text to Speech and Speech to Text services. If your customer asks to speak to a person, the phone integration can transfer the call to an agent.

## Updating and managing channels

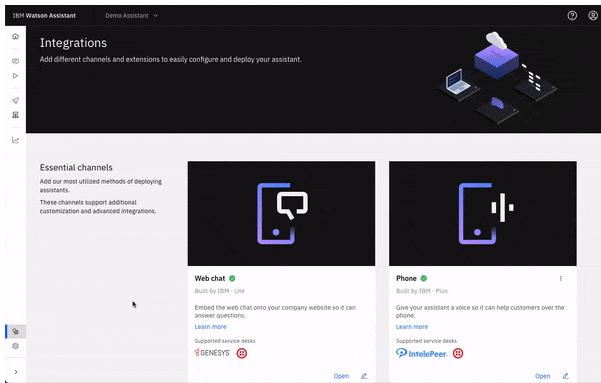
Each channel has specific settings that you can adjust to adapt the end experience for your user. You can edit these settings by selecting the channel in an environment, or in **Integrations**.

If you make an update to a channel in the draft environment, the same channel in live environment is not affected in the live environment. Similarly, if you make an update to a channel in the live environment, the same channel in draft environment is not affected. If you select a channel from the **Integrations** page, you are asked to select which environment you are editing.

For more information about editing your web chat integration, see [Basic web chat configuration](#).

## Deleting channels

To delete a channel, go to the **Integrations** page and use the overflow menu on the integration:



If you deployed your assistant to a channel, then deleting the channel does not remove the assistant from the channel. For example, if you deploy web chat, you paste the JavaScript snippet into the HTML header of your website. Deleting your channel disconnects your content from the customer experience that is shown on your website. Be sure to remove the JavaScript snippet before you delete the channel.

Deleting a channel is final and deletes the channel from all environments. Think twice before you delete an integration because each new integration that you might add starts with default settings. If you delete your channel and then decide you want it back in a previous state, you must rebuild the channel after you add it back.

## **Adding integrations**

Add integrations to your assistant so that you can publish your bot to the channels where your customers go for help.

To deploy an assistant to customers, a channel integration must be added. By default, a web chat integration is created, allowing an assistant to be embedded in a website. Other channel integrations are available in the **Integrations** catalog. For more information about deploying to your website, see [Adding the web chat to your website](#).



# Integrations

Add different channels and extensions to easily configure and deploy your assistant.

## Essential channels

Add our most utilized methods of deploying assistants.

These channels support additional customization and advanced integrations.



### Web chat

Built by IBM · Lite

Embed the web chat to answer questions. [Learn more](#)

When you add an integration, that integration is added to both the draft and live environments, or to all your environments if you are using [multiple environments](#). Test content and integrations before you deploy your assistant to customers. For more

information about adding integrations to your assistant, see [Adding integrations](#). After a live channel is added and configured, it is ready to deploy your assistant on its corresponding platform.

## Add an integration

Follow these steps to add integrations to your assistant:

1. Go to the **Integrations** page by clicking the integrations icon (⌘) in the left menu.
2. Scroll to see available integrations.

**Why do I have the *Web chat* integration?** This integration is provisioned and added automatically to your first assistant only.

Watson Assistant comes with the web chat integration, which is an engaging and fully extensible front-end client that can be added to your website in minutes. It is designed to handle advanced conversational scenarios, including rich responses, such as images, video, and iframes, suggestions when the user gets stuck, and live agent escalation.

3. For any integration that you want to add, click **Add**. The options include:

- [Web chat](#)
- [Phone](#)
- [Facebook Messenger](#)
- [Microsoft Teams](#)
- [Slack](#)
- [SMS](#)
- [WhatsApp with Twilio](#)
- [Search](#)

You can also set up live agent integrations by going to the **Live agent** tab from the **Web chat** tile. The options include:

### Web chat live agent integrations

- [Web chat with Salesforce support](#)
- [Web chat with Zendesk support](#)

### Web chat live agent reference implementations

- [Genesys Cloud](#)
- [NICE inContact](#)
- [Twilio Flex](#)
- [Kustomer](#)
- [Bring your own \(starter kit\)](#)

### Phone live agent integrations

- [Phone with Genesys Cloud](#)
- [Phone with NICE CXone](#)
- [Phone with Twilio Flex](#)
- [Bring your own live agent integration](#)

4. Follow instructions provided on the screen to complete the integration process.



**Note:** Plus For environments where private endpoints are in use, keep in mind that these integrations send traffic over the internet.

## How live agent integrations work

Watch a 4-minute video about integrating your assistant with a live agent integration, such as Zendesk:



[View video: Zendesk Integration: Watson Assistant](#)

To learn about how live agent integrations with your assistant can benefit your business, read [Customer Service Just Got Smarter](#).

## Integrating with Facebook Messenger

IBM Cloud

Facebook Messenger is a messaging application that helps businesses and customers communicate directly with one another.

After you create an action, you can integrate your assistant with Facebook Messenger.

**⚠ Important:** There is currently no mechanism for identifying users who interact with the assistant through Facebook Messenger, which means that there is no way to identify or delete data associated with a specific user. Do not use this integration method for deployments that must be GDPR compliant.

1. Go to the **Integrations** page by clicking the integrations icon (⚙) in the left menu.
2. Click **Add** on the **Facebook Messenger** tile.
3. Click **Confirm**.
4. Follow the instructions that are provided on the screen to complete the integration process.

## Action considerations

The rich responses that you add to the action are displayed in a Facebook app as expected, with the following exceptions:

- **Connect to live agent:** This response type is ignored.
- **Image:** This response type embeds an image in the response. A title and description are not displayed before the image, whether or not you specify them.
- **Option:** This response type shows a list of options that the user can choose from.
  - A description is not displayed, whether you specify one or not.
  - After a user clicks one of the buttons, the button choices disappear and are replaced by the user input that is generated by the user's choice. If the assistant or the user enters new input, then the button-generated input disappears. Therefore, if you include multiple response types in a single response, position the option response type last. Otherwise, content from subsequent responses, such as text from a text response type, will replace the button-generated text.
  - The title is automatically taken from the text of the relevant step of the action where options are listed.

## Chatting with the assistant

To start a chat with the assistant, complete the following steps:

1. Open Facebook Messenger.
2. Type the name of the page you created earlier.
3. After the page comes up, click it, and then start chatting with the assistant.

The welcome action is not processed by the Facebook Messenger integration. The welcome message is not displayed in Facebook Messenger like it is in the assistant preview. It is not triggered from here because nodes with the `welcome` special condition are skipped in action flows that are started by users. Facebook Messenger waits for the user to initiate the conversation.

The action flow for the current session is restarted after 60 minutes of inactivity (5 minutes for Lite and Standard plans). This means that if a user stops interacting with the assistant, after 60 (or 5) minutes, any context variable values that were set during the previous conversation are set to null or back to their default values.



**Note:** Only the page administrator can interact with the Facebook Messenger chatbot until after it is approved by Facebook. After the chatbot is approved by Facebook, any page visitor can interact with the chatbot.

## Integrating with Microsoft Teams

IBM Cloud

Add a chatbot to Microsoft Teams to create and customize a productive hub where content, tools, and people come together to chat, meet, and collaborate.

After you [create an action](#), you can use this integration to connect your assistant with Microsoft Teams.

### **Before you begin**

Sign up for a Microsoft 365 Developer Administrator email address, if you don't have one:

1. Go to the [Microsoft Dev Center](#).
2. Click **Join now** to become a member of the Microsoft 365 Developer program.
3. On the Dashboard, click **Set up E5 subscription**, and select an **Instant** or a **Configurable** sandbox.
4. Copy the email address listed under **Administrator**. Your admin credentials will be required at several points of setup.

### **Adding the Microsoft Teams integration**

1. Go to the **Integrations** page by clicking the integrations icon (⊕) in the left menu.
2. Click **Add** on the **Microsoft Teams** tile.
3. Click **Confirm**. The **Get Started** page provides a summary of the setup process.
4. Click **Next** to begin app registration.

### **App registration**

1. Go to the [Microsoft Azure portal](#), and log in with your admin credentials.
2. On the **App registrations** page, click **New registration**.
3. On the **Register an application** page, enter a name, select the multi-tenant option that applies to your app, and then click **Register**.

4. Copy the application ID from the **Overview** page of your app, and paste it into the **App registration** field of your {site.data.keyword.conversationshort} Microsoft Teams integration.
5. On the same Microsoft Azure **Overview** page, click the hyperlink **Add a certificate or secret** next to Client Credentials.
6. On the **Certificates & secrets** page for token creation, click **New client secret**. Enter a description and then select **Recommended 180 days**. Click **Add**.
7. Copy the string under **Value** and paste into **Client secret value** on the **App registration** page of your {site.data.keyword.conversationshort} Microsoft Teams integration. Note: You *must* generate a new value before the current one expires on day 180.
8. Click **Next** to create your bot.

## Create your bot

1. Go to the [Microsoft Bot Framework developer portal](#), and log in with your admin credentials.
2. On the **Tell us about your bot** page, complete your bot profile.
3. Copy the generated endpoint from the **Create your bot** page of your {site.data.keyword.conversationshort} Microsoft Teams integration and paste into the **Messaging endpoint** field of the **Configuration** section.
4. Select **Multi-Tenant** as the app type.
5. Copy and paste your app ID, and then click **Register**.
6. On the **Connect to channels** page, click **Configure Microsoft Teams channel** in the **Add a featured channel** section.
7. On the **Configure Microsoft Teams** page, specify options in the **Messaging**, **Calling**, and **Publish** tabs that fit your bot needs, and then click **Save**.
8. In your {site.data.keyword.conversationshort} Microsoft Teams integration, click **Next** to create your Teams app.

## Create your Teams app

1. Go to the [Microsoft Teams Developer Portal](#), and log in with your admin credentials.
2. On the **Apps** tab, click **New App**.
3. Enter a name, and click **Add**.
4. On the **Basic information** page, enter app names, app ID, descriptions, developer information, and app URLs, and then copy and paste your app ID into **Application (client) ID**. Click **Save**.
5. In the **Configure** section, select **App features**, and then **Bot**.
6. On the **Identify your bot** page, select your bot.
7. In the **Select the scope in which people can use this command** section, select **Personal**, **Team**, and **Group Chat**.
8. Click **Save**, but keep the window open.
9. In your {site.data.keyword.conversationshort} Microsoft Teams integration, click **Finish**.
10. Click **Publish** to publish your bot.

## Publishing your Teams app

1. In the Microsoft Teams Developer Portal window where you created and saved your Teams app, click **Publish** to publish your bot.
2. Click **Download the app package**.
3. Go to [Microsoft Teams](#), and log in with your admin credentials.
4. Click **Apps** in the sidebar menu, and then click **Manage your apps** and **Upload an app**.
5. Select **Upload a custom app** and specify the app package .zip file you downloaded.
6. Click **Add** to finish.
7. Test your actions and responses in the **Chat** section of your Teams app.

## Response types

These [response types](#) are supported and displayed as expected when your assistant is deployed for Microsoft Teams direct messages, channels, or group chats.

- date
- image
- option
- suggestion
- text

## Integrating with Slack

---

IBM Cloud

Slack is a cloud-based messaging application that helps people collaborate with one another.

After you create an action, you can integrate your assistant with Slack.

When integrated, depending on the events that you configure the assistant to support, your assistant can respond to questions that are asked in direct messages or in channels where the assistant is directly mentioned.

## Adding the Slack integration

1. Go to the **Integrations** page by clicking the integrations icon (⊕) in the left menu.
2. Click **Add** on the **Slack** tile.
3. Click **Confirm**.
4. You need to have a Slack app to connect to.

If you don't have a Slack app, create one now. See [Starting with Slack apps](#).

5. Go to the [Your Apps](#) page on the Slack website, and then click the app you want to use.

**Tip:** Open the Slack app in a new browser tab, so you can easily switch back and forth between the Slack app settings page and Watson Assistant Slack integration configuration page.

6. From the settings page for your Slack app, open the **App Home** page.

7. Add access scopes for your Slack app.

The button label might be *Review Scopes to Add* or *Update scopes* depending on whether you are creating a new app or editing an app that you created before February 2020.

 **Note:** The method for Slack access changed. For more information about it, read the [Slack blog post](#) about it.

8. Assign bot token scopes to your Slack app. At a minimum, apply the following scopes:

- `app_mentions:read`
- `chat:write`
- `im:history`
- `im:read`
- `im:write`

9. Click *Install App to Workspace*, and then allow the installation when prompted.

If you are editing scopes for an existing application, reinstall it.

10. From the Slack settings App Home page, enable the *Always Show My Bot As Online* setting.

11. Go to the *OAuth and Permissions* page in Slack, copy the *Bot User OAuth Access Token*.

12. From the Watson Assistant Slack integration configuration page, paste the token that you copied in the previous step into both the **OAuth access token** and **Bot user OAuth access token** fields.

13. On the Slack app settings page, go to the *Basic Information* page, and then find the *App Credentials* section. Copy the app credential verification token.

14. From the Watson Assistant Slack integration configuration page, paste the verification token that you copied in the previous step into the **Verification token** field.

15. Click **Generate request URL**, and then copy the generated request URL.

16. Return to the Slack app settings page. Open the *Event Subscriptions* page, and then turn on *Enable Events*. Paste the request URL that you copied in the previous step into the field.

17. On the *Event Subscriptions* page in Slack, find the *Subscribe to Bot Events* section. Click *Add Bot User Event*, and then select the event types you want to subscribe to. You must select at least one of the following types:

- `message.im`: Listens for message events that are posted in a direct message channel.
- `app_mention`: Listens for only message events that mention your app or bot.

 **Note:** Choose the `app_mention` entry in normal font, *not* the `app_mention` entry that is in bold font.

18. Click *Save Changes*.

19. Optional: To add support for showing buttons, menus, and disambiguation options in the Slack app, go to the *Interactive Components* tab and enable the feature. Paste your request URL in the provided text entry field, and then click *Enable Interactive Components*.

## Action considerations

The rich responses that you add to an action are displayed in a Slack channel as expected, with the following exceptions:

- **Connect to live agent:** This response type is ignored.
- **Option:** This response type shows a list of options that the user can choose from.
  - After a user clicks one of the options, the choices disappear and are replaced by the user input that is generated by the user's choice. If you include multiple response types in a single response, position the option response type last. Otherwise, the output might contain a mix of responses and user inputs that can confuse the user.
  - If the options are displayed in a drop-down list, then each option value must be 75 characters or fewer in length. When a list includes 5 or more options, it is displayed in a drop-down list automatically.

## Chatting with the assistant

To start a chat with the assistant, complete the following steps:

1. Open Slack, and go to the workspace associated with your app.
2. Click the application that you created from the Apps section.
3. Chat with the assistant.

The welcome action is not processed by the Slack integration. The welcome message is not displayed in the Slack channel like it is in the assistant preview. It is not triggered from here because nodes with the `welcome` special condition are skipped in action flows that are started by users. Slack waits for the user to initiate the conversation.

The action flow for the current session is restarted after 60 minutes of inactivity (5 minutes for Lite and Standard plans). This means that if a user stops interacting with the assistant, after 60 (or 5) minutes, any context variable values that were set during the previous conversation are set to null or back to their default values.

## Integrating with SMS

---

IBM Cloud

Add a text messaging integration so your assistant can exchange messages with your customers.

The Short Messaging Service (SMS) supports text-only messages. Typically, SMS restricts the text message length to 160 characters. The Multimedia Messaging Service (MMS) supports sending images and text messages that are over 160 characters in length. When you create a phone number with Twilio, MMS message support is included automatically.

Customers send text messages to your hosted phone number. Twilio uses a messaging webhook that you set up to send a POST request with the text message body to your assistant. Each response from the assistant is sent back to Twilio to be converted to an outbound SMS message that is sent to the customer. The responses are sent to the SMS Provider's API for processing. You provide your SMS Provider's authentication token information, which serve as your API access credentials.

Refer to the following sections to set up the integration for your SMS provider:

- [Integrating SMS with Twilio](#)

If you want your assistant to be able to switch between voice and text during a customer interaction, enable both the phone and text messaging integrations. The integrations do not need to use the same third-party service provider. For more information, see [Integrating with phone](#).

## **Integrating SMS with Twilio**

### **Before you begin**

If you don't have a text messaging phone number, set up an *SMS with Twilio* account and get a phone number.

1. Go to the [Twilio website](#).
2. Create an account or start a free trial.
3. From the **Develop** tab in the sidebar, click **Phone Numbers**. If **Phone Numbers** is not present, go to the Search Bar at the top and search for 'Phone Numbers', then select **Buy a number**.
4. Follow the instructions to **Buy a number**.

When you get a Twilio phone number, it supports voice, SMS, and MMS automatically. Your new phone number is listed as an active number.

**Tip:** Keep the Twilio web page open in a web browser tab so you can refer to it again later. You can also pin **Phone Numbers** to the sidebar.

## Set up the integration

To set up the integration, complete the following steps:

1. Go to the **Integrations** page by clicking the integrations icon (⌘) in the left menu.
  2. Click **Add** on the *SMS with Twilio* tile.
  3. Click **Add**.
  4. From the Twilio site, click on your account name in the upper left menu to go to your account dashboard.
- Copy the following values and store them temporarily, so you can paste them into the *SMS with Twilio* integration setup page in the upcoming steps.
- Account SID
  - Auth token
5. Return to the *SMS with Twilio* integration setup page. Click **Next** to go to Step 1 of your *SMS with Twilio* integration setup.
  6. Enter your **Account SID** information. Click **Next** to go to Step 2 of your *SMS with Twilio* integration setup.
  7. Enter your **Auth token** information. Click **Next** to go to Step 3 of your *SMS with Twilio* integration setup.
  8. **Optional:** Enter the phone number that your Twilio account uses for SMS integration. The webhook URI is used to transfer messages, but if you add your phone number in this optional field, you can easily refer to it later. Click **Next** to go to Step 4 of your *SMS with Twilio* integration setup.
  9. Copy the value from the **Webhook URI** field.

You will add this URI to the webhook configuration in Twilio. If you want to support more than one phone number, you must add the URI to the webhook for each phone number separately.

10. Go to your Twilio account web page. From the **Develop** tab in the sidebar, click **Phone Numbers > Manage > Active numbers**.
11. From the **Active Numbers** page, click one of your phone numbers.
12. Scroll to the **Messaging** section, and then find the **Webhook** field that defines what to do when *A message comes in*.

Paste the value that you copied from the **Webhook URI** field into it.

13. If you want to support multiple phone numbers, repeat the previous step for each phone number that you want to use.

14. Click **Save**.

15. From the **Develop** tab in the sidebar, click **Messaging > Settings > Geo permissions**. If **Messaging** is not present, go to the Search Bar at the top and search for 'Messaging', then select **SMS Geographic Permissions**.

16. From the **Messaging Geographic Permissions** page, select the country codes of the phone numbers that can text your Twilio number. By default, no country codes are allowed to text your Twilio number.

17. Return to the *SMS with Twilio* integration setup page. Click **Finish**.

## SMS Advanced configuration options

The **Advanced options** tab is available after you set up the SMS integration. Click the **Advanced options** tab to make any of the following customizations to the messaging behavior:

- **Initiate conversation from inbound messages:** Disable this option if you want to limit messaging support to allow messages that are sent in the context of an ongoing phone integration conversation only, and not allow customers to start a message exchange with the assistant outside of a phone call.
- **Default failure message:** Add a message to send to the customer if the SMS connection fails.
- **Base URL:** This URL is the REST API endpoint for the SMS service you are using.

## Optimize your actions for messaging

For the best customer experience, design your actions with the capabilities of the SMS integration in mind:

- Do not include HTML elements in your text responses.
- The SMS integration does not support chat transfers that are initiated with the `connect_to_agent` response type.
- **Image, Audio, Video** response types allow sending a message containing media. A title and description are sent along with the attachment. Note that depending on the carrier and device of the end user these messages may not be successfully received. For a list of the supported content types for Twilio, see [Twilio: Accepted Content Types for Media](#).

For more information on these response types, see [Response types reference](#).

If you want to use the same action for an assistant that you deploy to many different platforms, add custom responses per integration type. You can add a conditioned response that tells the assistant to show the response only when the SMS integration is being used.

For reference documentation, see [SMS integration reference](#).

## Troubleshooting

Find solutions to problems that you might encounter while using the integration.

- If you get a *Forbidden* message, it means the phone number that you specified when you configured the integration cannot be verified. Make sure the number fully matches the SMS phone number.

## Migrating from Voice Agent with Watson

If you created an IBM® Voice Agent with Watson™ service instance in IBM Cloud to enable customers to exchange text messages with an assistant, use the SMS integration instead.

The SMS integration provides a more seamless integration with your assistant and supports as many phone numbers as needed.

However, the integration currently does not support the following functions:

- Starting an SMS-only interaction with an outgoing text
- Configuring backup locations
- Reviewing the usage summary page. Use IBM Log Analysis instead.

To migrate from Voice Agent with Watson to the Watson Assistant SMS integration, complete the following step:

1. Do one of the following things:

- If your Voice Agent with Watson service instance uses an SMS service provider other than Twilio, you cannot continue to use it. You must create an SMS account with Twilio first. Complete the [Before you begin - Twilio](#) steps to create the account. Next, set up the integration.
- If your Voice Agent with Watson service instance uses Twilio as its SMS provider, you can go directly to [Set up the integration - Twilio](#).

## Integrating with WhatsApp

---

IBM Cloud

Integrate with WhatsApp messaging so your assistant can exchange messages with your customers where they are.

Many customers use WhatsApp because it provides fast, simple, secure messaging for free, and is available on phones all over the world. WhatsApp uses the phone Internet connection to send messages so customers can avoid SMS fees.

This integration creates a connection between your assistant and WhatsApp by using Twilio as a provider.

### Before you begin

If you don't have one, set up a Twilio messaging account and get a phone number.

1. Go to the [Twilio website](#).
2. Create an account.
3. From the **Develop** tab, click **Phone numbers**.
4. Follow the instructions to get a phone number.

When you get a Twilio phone number, it supports voice, SMS, and MMS automatically. Your new phone number is listed as an active number.

**Tip:** Keep the Twilio web page open in a web browser tab so you can refer to it again later.

### Ask WhatsApp for permission to enable your Twilio number for WhatsApp

WhatsApp has a rigorous process that they use to review all businesses that want to interact with customers over their network. WhatsApp, which is owned by Facebook, requires that you register your business with the Facebook business directory.

1. To register, go to the [Facebook Business Manager](#) website, and click **Create account**. Follow the instructions to create an account.
2. Make a note of your Facebook Business Manager ID. You will need this in the next step.

3. Submit the *Request to enable your Twilio numbers for WhatsApp* form from the [Twilio API for WhatsApp](#) web page.

Tips for specifying the following values:

- **Phone Number:** Specify the Twilio phone number that you created earlier.

Consider provisioning more than one phone number and going through the process of getting permission for the numbers in parallel. If your number was used by a different business previously (because Twilio assigned you a number that was used before, for example), WhatsApp will reject it.

- **Are you working with an ISV:** Select **No**.
- **Twilio Account SID:** From the Twilio site, click the home icon to go to your project dashboard to find the SID.
- **Facebook Business Manager ID:** Add the ID for the account that you created in the previous step.

4. Click **Request Now**.

Give WhatsApp time to evaluate and approve your request. It can take up to 7 days for your request to be approved.

## Set up the integration

To set up the integration, complete the following steps:

1. Go to the **Integrations** page by clicking the integrations icon (⌘) in the left menu.
  2. Click **Add** on the *WhatsApp with Twilio* tile.
  3. Click **Confirm**.
  4. From the Twilio site, click on your account name in the upper left menu to go to your account dashboard.
- Copy the following values and store them temporarily, so you can paste them into the *WhatsApp with Twilio* integration setup page in the upcoming steps.
- Account SID
  - Auth token
5. Return to the *WhatsApp with Twilio* integration setup page. Click **Next** to go to Step 1 of your *WhatsApp with Twilio* integration setup.
  6. Enter your **Account SID** information. Click **Next** to go to Step 2 of your *WhatsApp with Twilio* integration setup.
  7. Enter your **Auth token** information. Click **Next** to go to Step 3 of your *WhatsApp with Twilio* integration setup.
  8. Copy the value from the **Webhook URI** field.
- You can use this webhook URI to test your integration in the following section.
9. Click **Finish**.

## Testing the integration

While you wait for WhatsApp to approve your request, you can test the integration by using the Twilio sandbox. With the sandbox, you can send and receive preapproved template messages to numbers that join your sandbox, using a shared Twilio test number.

Do not use the Twilio sandbox in production. Sandbox sessions expire after 3 days.

1. To create a sandbox, go to the [Twilio Try WhatsApp](#) web page. An *Activate your sandbox* prompt is displayed. Agree to have a sandbox created, and confirm your choice.
2. Follow the instructions to create the sandbox.
3. Connect to the sandbox by sending a WhatsApp message from your device to the sandbox phone number.
4. From the **Develop** tab, click **Messaging > Settings > WhatsApp sandbox settings**.
5. In the **Sandbox Configuration** section, paste the webhook URI that you copied earlier into the *When a message comes in* field. Click **Save**.
6. You can test the integration by sending a message from WhatsApp to the shared phone number that is assigned to your Twilio sandbox.

## Finish the product integration

After WhatsApp grants permission for your Twilio phone number or number to access the WhatsApp network, update the integration to use your dedicated Twilio phone number instead of the sandbox number.

1. From the *WhatsApp with Twilio* integration setup page, scroll to the **Webhook** section of the **Basic setup** tab. Copy the value from the **WhatsApp webhook** field.
2. Go to your Twilio account web page and add the webhook that you copied to the Twilio configuration to complete the connection to the WhatsApp integration in Twilio.

## Give customers fast access to your assistant

You can add an icon to your web page that customers can click to start a conversation over WhatsApp with your assistant.

To add an icon to your web page, complete the following steps:

1. From the *WhatsApp with Twilio* integration setup page, click the **Click to chat** tab.
  2. In the **Pre-filled message** field, add text that you want WhatsApp to send to your assistant on the customer's behalf to get the conversation started.
- Specify a message that you know your assistant can answer in a useful way.
3. Copy the **Embed link** and add it to your web page. Consider adding text in front of the icon that explains what the icon does. For example, you might add a `<span>` HTML tag in front of the icon's `<span>` element that says `Have a question? Ask Watson Assistant for help.`

When a user clicks the icon on your web page, it opens a WhatsApp messaging session that is connected to your assistant, and adds the text you specify into the user's text field, ready to be submitted.

## Action considerations

For the best customer experience, design your actions with the capabilities of the WhatsApp integration in mind:

- A text response that contains more than 1,600 characters is broken up into multiple responses.
- Do not include HTML elements in your text responses.
- The WhatsApp with Twilio integration does not support chat transfers that are initiated with the *Connect to agent* response type.

- If you use Markdown syntax, see the *Supported Markdown syntax* table.
- To include a hypertext link in a text response, specify the URL directly. Do not use markdown syntax for links. For example, specify Contact us at <https://www.ibm.com>.

Format	Syntax	Example
Italics	<code>We're talking about _practice_.</code>	We're talking about <i>practice</i> .
Bold	<code>There's *no* crying in baseball.</code>	There's <b>no</b> crying in baseball.

#### Supported Markdown syntax

## Integrating with a custom client app

### IBM Cloud

If the available integration channels do not meet your needs, you can build your own client application as the interface between the assistant and your customers.

For more information, see [Building a custom client using the API](#).

# Improving your assistant

## Use analytics to review your entire assistant at a glance

The **Analyze** page provides a summary of the interactions between users and your assistant.

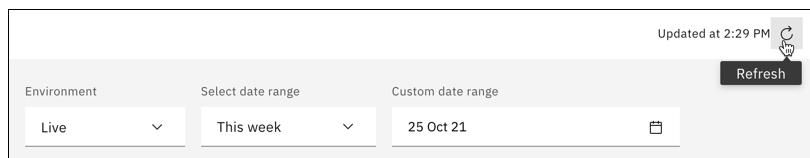
Analytics help you to understand the following things:

- *What do customers want help with today?*
- *Is your assistant understanding and addressing customer needs?*
- *How can you make your assistant better?*

To see analytics information, select **Analyze** in the navigation bar.

### Choosing the environment and date range

To get started, choose the [environment](#) and date range you want to analyze. All charts and cards reflect data based on the environment and the date range you select. When you change the environment or the date range, the charts and cards on the page update to reflect the new date range. You can also use **Refresh** to ensure the latest data is shown.



### Unique users, conversations, and user requests

These three traffic metrics -- *unique users*, *conversations*, and *user requests* -- provide you with data about the volume of user engagements with your assistant.



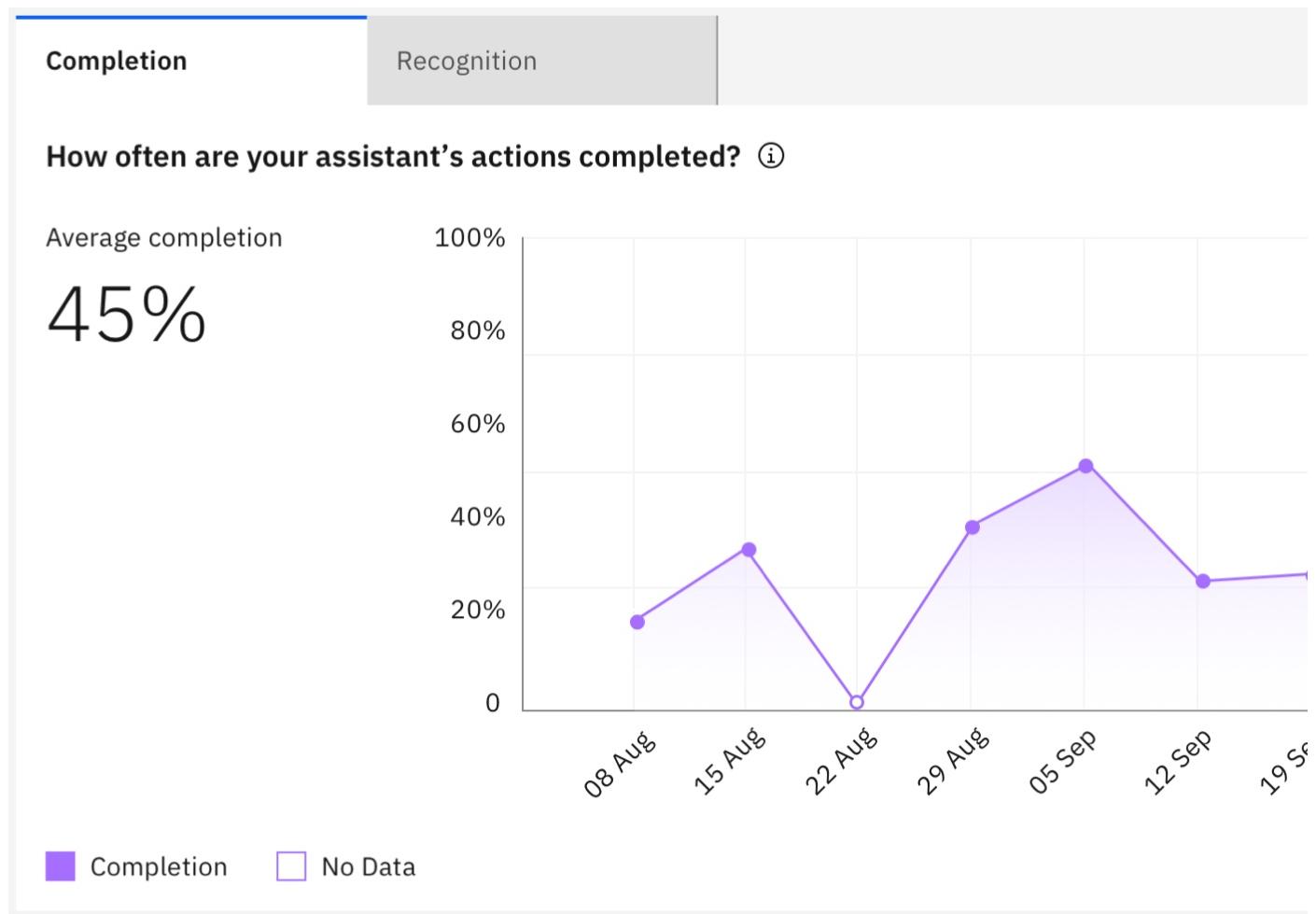
*Unique user* is anyone who interacts with your assistant. *User ID* identifies each user, using a unique label to track the level of service usage. A unique user can have multiple conversations, but a conversation never has more than one unique user.

*Conversation* is a set of messages consisting of the messages that an individual user sends to your assistant, and the messages your assistant sends back. Conversations can have multiple requests within a single conversation, but a single request doesn't span more than 1 conversation.

*Request* is a root-level utterance, such as an initial question or request, that signals the start of a specific flow. A user can initiate multiple requests. Requests are meant to represent the core concepts or topics your users are asking about. A request can have multiple steps within it, for example `I want to order a pizza` is a request. `Delivery/takeout`, `Small/Medium/Large`, `Cheese/Pepperoni/Mushrooms/Peppers` are all steps within the request of ordering a pizza.

## Completion and recognition

The *completion* and *recognition* charts provide information about the actions in your assistant.



### Completion

*Completion* is the measurement of how often users are able to successfully get through all steps of an action.

Your assistant measures when someone reaches the final step of an action. The completion chart provides an overview of all the actions you have built and how many of these are being completed or not.

Completion is only applicable when a user question or request matched to an action, and the action starts.

One action can be triggered multiple times, so to better understand individual action performance, click `action completion` to understand each action in more detail.

### Recognition

*Recognition* is the measurement of how many requests are being recognized by the assistant and routed into starting an action.

The recognition chart provides you with a view into how many requests are matched to actions. This helps you to understand where you may have content gaps, where you might want to build new actions, or how existing actions aren't matching properly to user requests.

Customer requests are considered unrecognized if:

- The request triggers the *No Action Matches* action
- The assistant asks a clarifying question and the customer chooses *None of the above*

To get more detail, click on *Unrecognized requests* to review the requests that are not being recognized by the assistant, so you can create new actions that address questions and issues that aren't being answered.

## Most frequent, least frequent, and least completed

These most/least cards help you to quickly identify specific actions that might need your attention. From these lists, you can click a specific action to do more analysis.

Most frequent actions		Least frequent actions	
08 Aug 2021 - 17 Oct 2021		08 Aug 2021 - 17 Oct 2021	
Actions	Number started	Actions	Number started
<a href="#">Pay bill</a>	955	<a href="#">Add to account</a>	1
<a href="#">Account information</a>	635	<a href="#">Mortgage payment</a>	2
<a href="#">Open account</a>	555	<a href="#">Close account</a>	3
<a href="#">Deposit check</a>	499	<a href="#">Budgeting plan</a>	5
<a href="#">Dispute a charge</a>	450	<a href="#">Withdraw money</a>	10

*Most frequent actions* shows the actions that have the most recognized requests matched with that action.

*Least frequent actions* is the exact opposite of *most frequent*. It can help you identify actions that possibly have poor training and are not matching well to actual requests from your users.

*Least completed actions* shows a list of actions with the lowest completion rate, by percentage. This measurement doesn't take traffic into account. This can be another good indication of actions where maybe the flow or content can be improved, and you can spend time investigating.

## Understand your most and least successful actions

The **Action completion** page of Watson Assistant provides an overview of how all your assistant's actions are doing. The page allows you to:

- Understand how well users are progressing through the action steps you have created
- Identify problem areas within actions
- Investigate why users are having issues where they might escalate to a live agent, start a new action, get stuck on a step, or stop the conversation

## Definition of completion

*Completion* measures how often within a given time period users reach the end step of an action.

## Reasons for completion

An action is considered complete when:

- A final (end) step is reached
- Search reached a final step
- The action called another action with the `End this action after the other action is completed` option, and the other action has completed
- Connect to agent transfer occurs according to the step response and without involving the [Fallback action](#)
- The last step of the action has been executed and there are no more steps to execute

## Reasons for incompletion

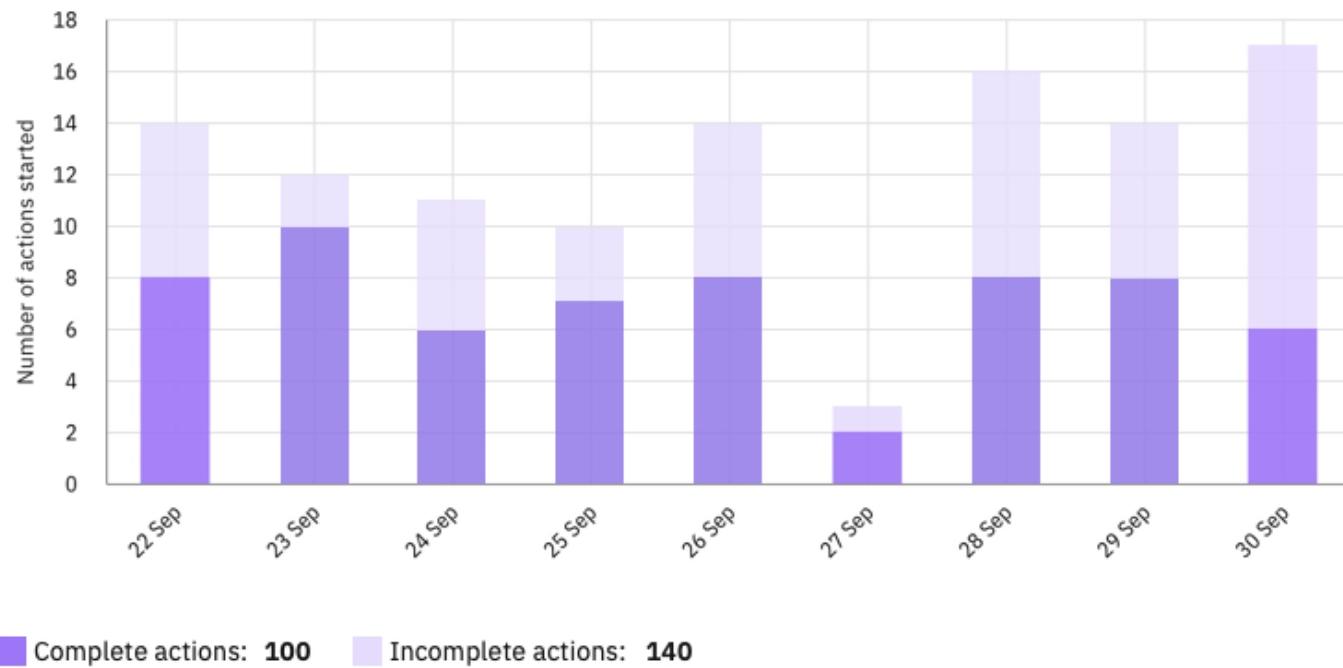
An action is considered incomplete for these reasons:

Reason	Description
Escalated to agent	The user explicitly asks to speak to someone, triggering the <a href="#">Fallback action</a> . For more information, see <a href="#">When your customer asks to speak to a live agent</a> . Or, the user chooses human agent escalation from the list of <a href="#">suggestions in web chat</a> .
Started a new action	The user changes the topic of the conversation, triggering another action, and either doesn't return to the original action or the other action is also incomplete.
Stuck on a step	Triggered during step validation where a user exceeds the maximum retries for the particular step. Default tries is set to 3, but you can change this setting. See <a href="#">Customizing validation for a response</a> for more information.
Abandoned	An action is considered abandoned if it was not completed after 1 hour of inactivity and doesn't meet the criteria for any other incompletion reason (escalated to agent, started a new action, or stuck on a step).

## Improving completion

To help you identify actions that need improvement, the **Action completion** page is organized into a **How often** chart and an **Actions** table. The **How often** chart shows either the percentage of complete actions or the number of complete and incomplete actions during a timeframe. Use the icon in the upper right of the chart to toggle between a line chart that shows the percentage of complete actions or a bar chart that shows the number of complete and incomplete actions. The **Actions** table provides number of requests, total incomplete, and completion percentage per action.

## How often are your recognized actions completed? ⓘ

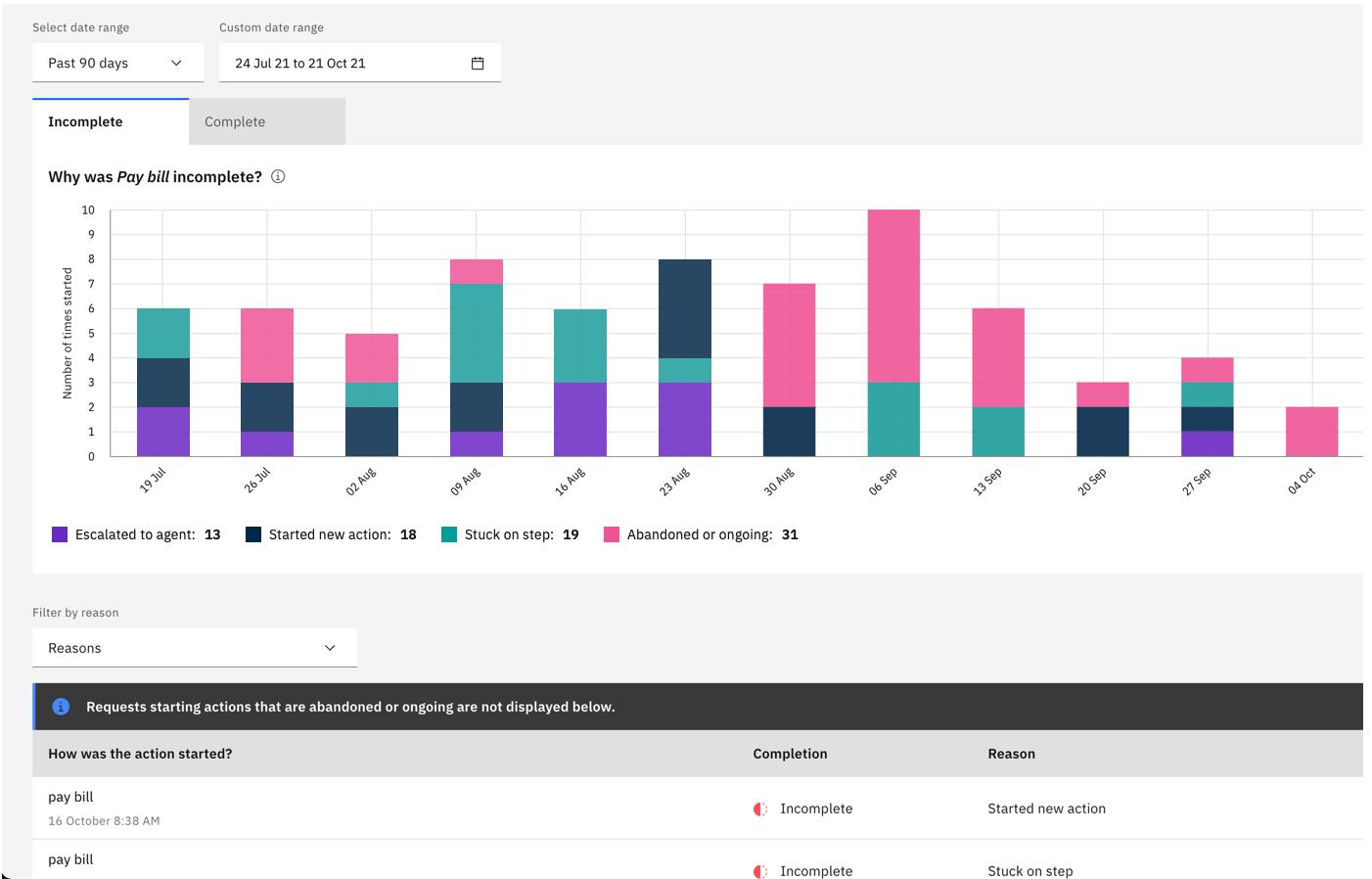


Actions	Times started
Pay bill	100
Deposit money	40
Withdraw Action	100

You can use the **Actions** table to focus on improving the completion of your actions. **Incomplete** results provide a focus area where you can make the biggest impact. The actions with the highest number of incompletions means these are the actions with the most users unable to get their questions answered or requests resolved.

To work on a specific action, click on the action name in the table. A page opens with completion details for that single action.

## Pay bill



The *Incomplete* and *Complete* tabs provide details by timeframe. The *Incomplete* tab bar chart shows data for the four incompleteness reasons. The *Complete* tab bar chart is organized by conversations either completed by the assistant or completed by a planned `connect to agent` response.

As you click on each tab, a table below the chart shows the list of either incomplete or complete requests. To explore individual requests in detail, you can click on each one. A panel opens showing the full back and forth between your customer and the assistant, including step interactions. The panel also provides a summary of how many requests there were, how many were recognized, whether search was initiated, and the duration of the conversation.

## Pay bill

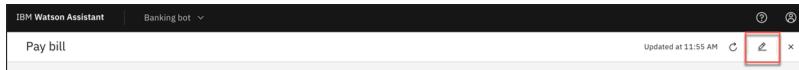
Filter by reason		Filter by keyword	
Reasons	▼	Keyword	Completion
Recognized requests <small> ⓘ</small>		Completion	Reason
how do I pay my bill online?	10 Feb at 11:55 AM	<span>Incomplete</span>	Stuck on
I need to pay my overdraft fees	10 Feb at 11:55 AM	<span>Incomplete</span>	Escalated
Pay credit card bill	10 Feb at 11:55 AM	<span>Incomplete</span>	Stuck on
I just overdrafted my checking account, i need to pay my bill immediately.	10 Feb at 11:55 AM	<span>Incomplete</span>	Escalated
I need to pay my credit card bill	10 Feb at 11:55 AM	<span>Incomplete</span>	End step
How do I pay my bills electronically?	10 Feb at 11:55 AM	<span>Incomplete</span>	Escalated
Pay my online bill statement	10 Feb at 11:55 AM	<span>Incomplete</span>	End step
Set up automatic bill payments	10 Feb at 11:55 AM	<span>Incomplete</span>	Started new
Online bill pay	10 Feb at 11:55 AM	<span>Incomplete</span>	End step
Pay bill now!	10 Feb at 11:55 AM	<span>Incomplete</span>	Started new

Requests per page: 100

1 – 100 of 1,228 requests

## Editing an action

After you decide on changes you want to make to improve the action, click the edit icon in the header for the single action, which opens your action for further work on its steps.



## Use unrecognized requests to get action recommendations

The **Recognition** page lets you analyze unrecognized requests. You can use this information to create new actions that address questions and issues that aren't being answered by your assistant.



**Note:** You can analyze unrecognized requests in English-language assistants only.

*Recognition* measures the requests within a given time period that are recognized and successfully routed to an action. Customer requests are considered unrecognized if:

- The request triggers the *No Action Matches* action
- The assistant asks a clarifying question and the customer chooses *None of the above*

## Viewing groups of unrecognized requests

You can view groups of unrecognized requests for the draft or live environment. Watson Assistant generates groups of similar unrecognized requests from the last 30 days so that you can decide whether to add the requests as example phrases to a new action.

Unrecognized requests <small>Data is from the past 30 days. <a href="#">Learn more</a></small>			
Group	Examples	Request count (362)	Percentage
unlock lmc account	"unlock lmc acct", "can you help to unlock my account", "please u...	38	10%
ok give minutes	"ok give me a minute please i will have a check", "ok give me a minut...	26	7%
systems ip address	"ok can you share me the systems ip address", "may i know your ope...	26	7%
not working dial	"since 2 days my desk phone is not working", "att dialer not working ...	26	7%
expense reimbursement application	"i am unable to get the expense reimbursement application to launc...	26	7%
deleted emails	"need to recover deleted emails deleted by accident", "how can i rec...	25	7%

The algorithm that generates groups considers several factors in shaping the groups:

- Unrecognized requests that have fewer than 2 or more than 35 significant words are removed from consideration. Common words such as *my* or *is*, or punctuation such as *?*, are not considered significant. Phrases that are too short or too long are usually not effective as example training phrases for your assistant.
- The unrecognized requests are compared to the latest version of your actions so that requests that would no longer be unrecognized with your latest actions version are filtered out.
- Groups for which the request count is less than 10 are excluded. But if this might result in less than 5 groups, the algorithm tries to produce groups that include more than 5 request counts until a total of 5 groups are produced.
- If you only have a small volume of data, the algorithm allows examples closer to the existing training data for grouping.

As a result of the algorithm:

- A list of groups might not always appear
- The groups might not include all unrecognized requests that you see on the **Conversations** page

Several events cause your groups to be refreshed using the latest data:

- The first time you visit the **Recognition** page for a specific environment, Watson Assistant generates groups
- If at least 1 day has passed since the groups were last generated, or if you edit your actions in your draft environment, the groups are refreshed when you return to the **Recognition** page
- You can refresh the groups using the **Refresh** icon

This table explains the details of the list of groups:

Column	Description
Group	A name is generated based on the example phrases in the requests.
Examples	Some of the unrecognized requests are shown as a preview.

Request count	The number of unrecognized requests in the group. You should focus on groups with a higher count of unrecognized requests.
Percentage	The percentage of unrecognized requests relative to the other groups. You should focus on groups with a higher percentage of unrecognized requests.

### Unrecognized request groups list

## Creating actions from unrecognized requests

To create actions based on unrecognized requests:

1. Click a group to open its page. Each unrecognized request is listed individually.

Column	Description
Unrecognized requests	Verbatim requests from customer conversations
Request count	The number of times the unrecognized request is included in this group
Related actions	Existing actions that you might modify to address the unrecognized request

### Unrecognized request group page

2. If Watson Assistant identifies additional similarities among the examples, you can click **Grouped by similarity** to further categorize the list.

The screenshot shows a 'Create new action' interface for an 'expense reimbursement application' group. It includes a 'Create a new action' button and a 'Grouped by similarity' option selected. The main area shows 'Unrecognized requests' with a count of 22, and a list of two examples: 'i can not launch expense reimbursement application' and 'hi i need help with my expense report'. The second example is highlighted with a blue border.

In a group named `expense reimbursement application`, requests might be further grouped by similarity. For example:

- Problems logging into or launching the expense reimbursement application
- Issues with expense reimbursement application crashing
- Requests for help or assistance with the expense reimbursement application

This can help you decide to add one or more actions that correspond to what you want to cover with your assistant, rather than adding all requests to a single action.

3. You can click to select unrecognized requests that you want to use as example phrases in a new action.

unlock lmc account

Updated at 11:59 PM

**Create new action**  
Select requests to start building a new action. Requests are used as example phrases.

View requests  Individually  Grouped by similarity

**Create a new action** +

Unrecognized requests	Request count (38)	Related actions
<input checked="" type="checkbox"/> "unlock lmc acct"	2	
<input checked="" type="checkbox"/> "unlock lmc account"	2	
<input checked="" type="checkbox"/> "please unlock my lmc account user i d is katiegau1i...	2	
<input checked="" type="checkbox"/> "unlock lmc account hello"	2	
<input checked="" type="checkbox"/> "can you help to unlocked my account"	2	

4. After you make your selections, click **Create new action**.
5. Enter a name for the action, or use the default, and then click **Apply**.
6. The action editor opens with each of the unrecognized requests included as an example phrase. You can now build an action to address these questions or issues.

unlock lmc account

**Customer starts with:**

unlock lmc acct

**Conversation steps**

**Customer starts with:**

Enter phrases that a customer types or says to start the conversation about a specific topic. These phrases determine the task, problem, or question your customer has.

The more phrases you enter, the better your assistant can recognize what the customer wants.

Enter phrases your customer might use to start this action Total: 5

Enter a phrase

can you help to unlocked my account	<input type="button" value="Delete"/>
unlock lmc account hello	<input type="button" value="Delete"/>
please unlock my lmc account user i d is kati	<input type="button" value="Delete"/>
unlock lmc account	<input type="button" value="Delete"/>
unlock lmc acct	<input type="button" value="Delete"/>

## Modifying existing actions

You can also use the unrecognized request groups to identify existing actions that you might want to modify. You can:

- Add unrecognized requests as example phrases to existing actions
- Move example phrases from existing actions and add them to any new actions you create based on unrecognized requests

If a group lists related actions, you might focus on modifying them to address the unrecognized request.

Unrecognized requests	Request count (57)	Related actions
<input type="checkbox"/> "lotus notes installation get frozen every time at 94"	2	
<input type="checkbox"/> "need to migrate lotus notes to 91"	1	need reinstall lotus, archive lotus notes, lotus notes reset
<input type="checkbox"/> "lotus notes 9 crashes"	1	lotus notes not - todd, help lotus notes, notes crashes opening
<input type="checkbox"/> "inotes working however lotus notes 9 still giving an expired certif...	1	help lotus notes, lotus notes not - todd, lotus notes ticket

## Downloading groups

You can download all group data or individual group data in a CSV file.

To download all group data, click the **Download groups** icon on the **Recognition** page.



To download data for an individual group, open the group, then click the **Download group** icon.



The CSV file includes this information:

Column	Description
Group	The name of the unrecognized requests group
Example	Each verbatim request in the group
Count	The number of times the unrecognized request is included in this group
Group Id	ID number for the group
Similarity Group Id	ID number for any further grouping by similarity, for example, 1, 2, and 3 if there are three similarity groups
Example Id	ID number for the example within the group or similarity group

[Download group CSV file](#)

## Review customer conversations

The **Conversations** page of Watson Assistant provides a history of conversations between users and a deployed assistant. You can use this history to improve how your assistants understand and respond to user requests.

Each timestamp represents a single conversation. The **Actions** column shows you how many actions, search queries, or unrecognized requests are included in that conversation. The **Requests** column includes the questions or requests the user entered that initiated an action, started a search query, or weren't recognized.



**Note:** If you have activated dialog in your assistant, the **Actions** column is replaced by a **Topics** column.

Environment	Select date range	Custom date range	Filter by actions	Filter by keyword or session ID
Draft	Past 2 weeks	05 Sep 22 to 13 Sep 22	Actions	Search
Conversations		Actions		Requests
09 Sep at 7:13 AM 812d9899-8792-4f20-9a7d-909a44b8...	name Greet customer		"register name" "f36b"	
09 Sep at 7:12 AM f36bfb48-b397-4be8-bd27-f4aa4471...	name		"register name"	

## Choosing the environment and time period

To get started, choose the [environment](#) and date range you want to analyze. All conversations reflect data based on the environment and the date range you select. When you change the environment or the date range, the conversations on the page update to reflect the new date range. You can also use **Refresh** ensure the latest data is shown.

The screenshot shows the top navigation bar of the Watson Assistant interface. It includes fields for 'Environment' (set to 'Draft'), 'Select date range' (set to 'Past 2 weeks'), 'Custom date range' (set to '05 Sep 22 to 13 Sep 22'), 'Filter by actions' (set to 'Actions'), and 'Filter by keyword or session ID' (set to 'Search'). A 'Refresh' button is located in the top right corner of the header.

## Filtering conversations

You can locate specific conversations by filtering the list of conversations. This lets you explore specific areas where your assistant might need improvement or updates to properly handle what your customers are asking about.

You can filter by:

- **Actions:** Select specific actions. You can choose one or more actions to review.
- **Keyword:** Search by session ID or for specific key terms, phrases, or words in the conversations. For more information about session IDs, see [session\\_id](#).
- **Recognition:** Choose between recognized or unrecognized user questions or requests.
- **Search:** Choose between requests that initiated a search or requests that produced no search results.

The Actions and Keyword filters always appear at the top of the page. To show the Recognition and Search filters, click the **Additional filters** icon.

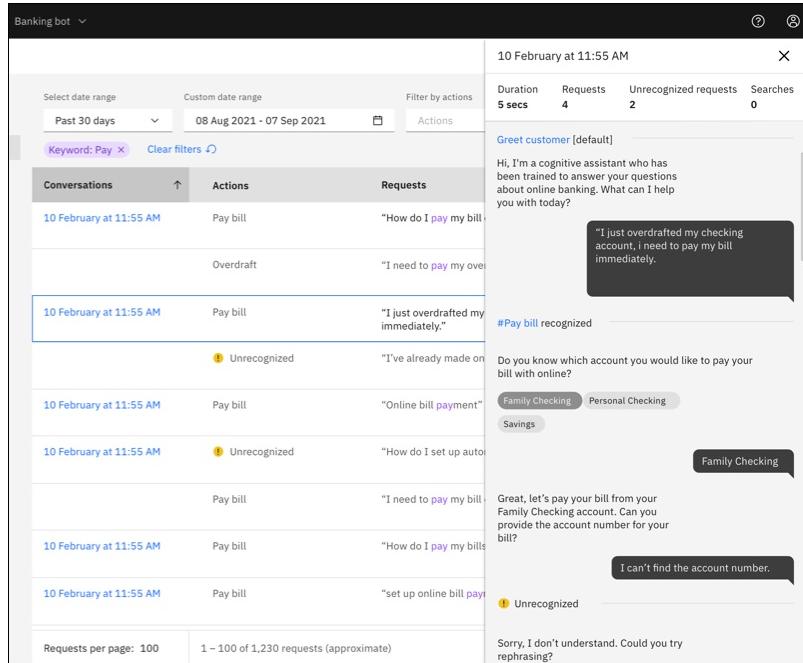
 **Note:** If you have activated dialog in your assistant, the **Actions** filter is replaced by a **Topics** filter.

The screenshot shows the top navigation bar with the 'Actions' filter replaced by a 'Topics' filter. Below the navigation bar, a 'Recognition' filter is expanded, showing dropdown menus for 'Search' and 'All'.

 **Note:** If you search for a specific session ID, enclose your search in quotation marks to ensure you receive an exact match on the full ID with its numbers and hyphen characters, for example: "9015ab9a-2e13-4627-ae33-4179b1125cb5".

## Exploring conversations in detail

To explore individual conversations in detail, you can click on any of the utterances or conversation time stamps. A panel opens showing the full back and forth between your customer and the assistant, including step interactions. The panel also provides a summary of how many requests there were, how many were recognized, whether search was initiated, and the duration of the conversation.



The screenshot shows the Watson Assistant interface with a 'Banking bot' selected. The main area displays a list of conversations. A specific conversation from '10 February at 11:55 AM' is expanded, showing a back-and-forth between the customer and the assistant. The customer's message 'I just overdrafted my checking account, I need to pay my bill immediately.' is highlighted with a dark background and white text. The assistant's response is also visible. At the bottom of the expanded panel, there are buttons for 'Family Checking', 'Personal Checking', and 'Savings', with 'Family Checking' being the active tab. The bottom left of the main interface shows a date range selector from '08 Aug 2021 - 07 Sep 2021' and a 'Actions' button. The bottom right shows summary statistics: Duration 5 secs, Requests 4, Unrecognized requests 2, and Searches 0.

## Sending events to Segment

IBM Cloud

You can use the Segment extension to send Watson Assistant events to Segment.

### Overview

With this extension, you can use [Segment](#) to capture and centralize data about your customers' behavior, including their interactions with your assistant. Events are sent from Watson Assistant to Segment, making them available to destinations such as data warehouses, raw data tools, and analytic tools.

For more information about the events that are sent to Segment, see [Segment event reference](#).

This feature is available only to Enterprise plan users.

### Adding the extension to the draft environment

To add the Segment extension to your assistant, follow these steps:

1. On the [Integrations](#) page, scroll to the **Extensions** section and find the tile for the Segment extension.
2. Click **Add**. Review the overview of the extension and click **Add** to configure it for your assistant.

**⚠️Important:** When you first add the Segment extension to an assistant, the configuration settings you provide are applied only to the draft environment. You must complete configuration for the draft environment before you can

add the extension in the live environment.

3. In the **Connect** step, click the link to log in to your Segment account in another browser tab.

If you do not already have a Segment account, click **Sign up for free account** to create one. Verify your email address and complete your profile to activate your account.

4. In the Segment web app, go to your workspace. In the **Sources** section, find the **Add IBM Watson Assistant Source** tile and click **Add Source**.

5. In the **Source Name** field, type a descriptive name for your Watson Assistant instance (for example, `Customer Care Assistant`). Click **Create Source**.

6. Click **Copy** to copy the generated key to the clipboard.

7. Go back to the Segment integration settings in the Watson Assistant interface. In the **Segment key** field, paste the key you copied from the Segment web app in the previous step. Click **Next**.

8. In the **Select events** step, review the list of events Watson Assistant can send to Segment.

Each row in the table shows the name of a supported event, along with a brief description.

9. Click the checkboxes to select the events you want to send to Segment. Click **Next**.

10. In the **Review & Confirm** step, review the configuration and click **Finish**.

The Segment extension is now connected to your assistant in the draft environment. Click **Close** to close the integration settings.

## Configuring the extension for the live environment

To configure the Segment extension for the live environment, follow these steps:

1. On the  **Integrations** page, scroll to the **Extensions** section and find the tile for the Segment extension.

2. Click **Open**. The **Open extension** window opens.

3. In the **Environment** field, select **Live**. Click **Confirm**.

4. Repeat the configuration process, specifying the values you want to use for the live environment.

 **Note:** If you are using multiple environments, follow the same steps to configure the extension for each environment.  
For more information, see [Adding and using multiple environments](#).

The Segment extension is now available in the environments you have configured, and events will be sent to the destinations configured in your Segment workspace. (For more information about the events that are sent to Segment, see [Segment event reference](#).)

# Administering your instance

## Administering your instance

---

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

## Managing access

---

If you need to collaborate with others on your assistants, you can quickly add users to your service instance from the **Manage** menu. Or to tailor specific access to your assistants, use the [Identity and Access Management \(IAM\) page](#) in IBM Cloud.

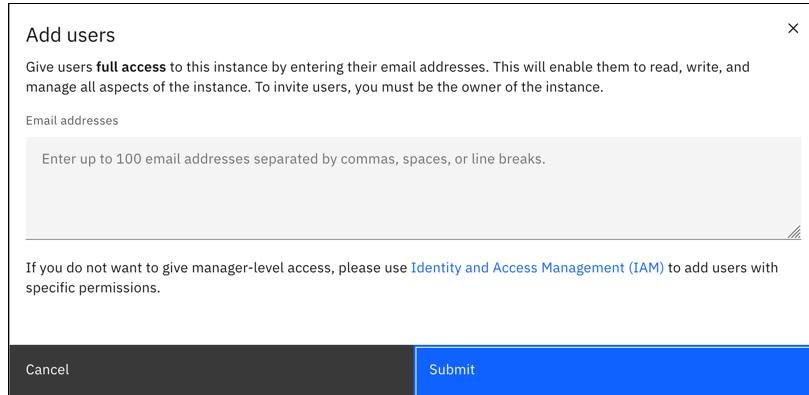
### Adding users from the Manage menu

In the new Watson Assistant, each assistant contains all the draft and live resolution methods (actions and search integration) and channels you add (such as web chat, Facebook, or Slack). The simplest way to provide access is to add users to your Watson Assistant service instance with manager access to all assistants. Users get all the privileges that they need to build and deploy any assistant.

To quickly add users with manager access to all assistants, complete the following steps:

1. Open the **Manage** menu. 
2. Click **Add users**.
3. Enter the email addresses of the users that you want to provide full access to. Separate email addresses with commas,

spaces, or line breaks.



Add users

Give users **full access** to this instance by entering their email addresses. This will enable them to read, write, and manage all aspects of the instance. To invite users, you must be the owner of the instance.

Email addresses

Enter up to 100 email addresses separated by commas, spaces, or line breaks.

If you do not want to give manager-level access, please use [Identity and Access Management \(IAM\)](#) to add users with specific permissions.

Cancel      Submit

Add users

**⚠ Important:** Adding users from this menu enables them to read, write, and manage all assistants in the service instance.

4. Click **Submit**.

After you click **Submit**, any user that you invite receives an email to access the instance. After they accept the invite, they can open the service instance and manage all assistants.

## Managing access with Identity and Access Management

Another way to add users to your assistants is using Identity and Access Management (IAM). If you want to add users, and you don't want them to have full Manager access, use IAM to add them. From IAM, you can also manage access roles of those users that are already added to your assistants.

## Opening Identity and Access Management

1. Open the **Manage** menu. 
2. Click **Manage users**.
3. In **Access and permissions**, click **Identity and Access Management** in step 2.

Review recommended roles 2. Invite users and apply roles. Go to the [Identity and Access Management](#) page in IBM Cloud, and start by clicking [Invite users](#)." data-bbox="266 675 762 795"/>

Manage users

Access and permissions

You can share this service instance and its assistants and skills with your colleagues by inviting them to the instance, and assigning them roles that govern permissions.

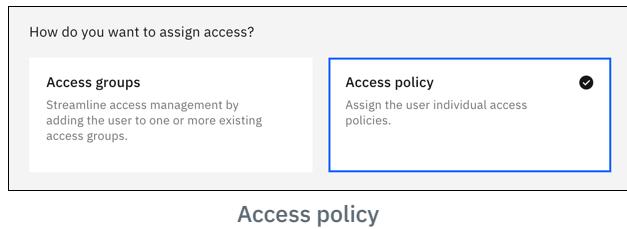
1. Choose the right roles to assign. [Review recommended roles](#)
2. Invite users and apply roles. Go to the [Identity and Access Management](#) page in IBM Cloud, and start by clicking [Invite users](#).

Access and permissions

## Adding users in Identity and Access Management

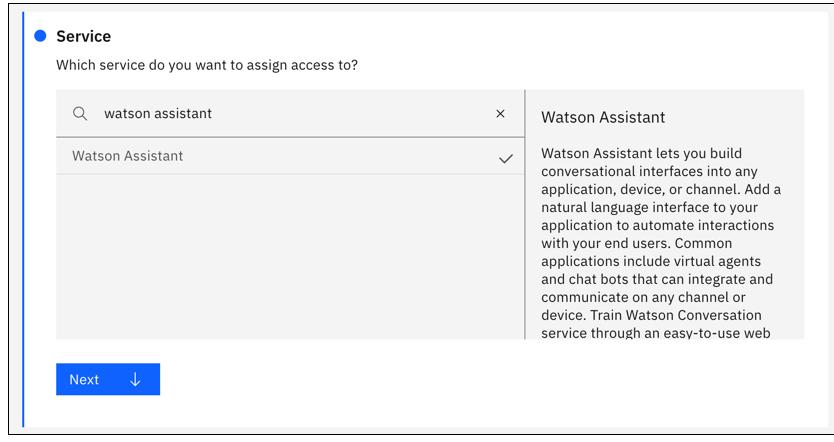
1. In IAM, click **Invite users**.
2. Enter the email address of the person who needs access.

3. In **How do you want to assign access?**, choose **Access policy**.



Access policy

4. In **Service**, choose **Watson Assistant**, then click **Next**.



Service

5. In **Resources**, choose either **All resources** or **Specific resources**.

If you choose **All resources**, the user can access all the instances of Watson Assistant in your account.

If you choose **Specific resources**, you can narrow access in **Attribute type**. With this setting, you might need to add multiple access policies for a user to grant the correct access. For more information, see [Example of limiting access to one assistant](#).

Choices include:

Resource attribute	Description
<b>type</b>	
Service Instance	Choose a specific service instance of Watson Assistant
Assistant, Environment, or Skill ID	Enter the ID value for the resource. Use <b>Assistant settings</b> to get the ID values for your assistant, environments, action skill, or dialog skill.
Resource Type	If you enter an ID value, choose Assistant ID, Environment ID, or Skill ID to identify the ID type
Region	Choose a specific region (for example, Dallas or London)
Resource Group	Enter or choose a resource group that you created

Attribute types

**Resources**

How do you want to scope the access?

All resources  
 Specific resources

Attribute type: Service Instance

Operator: string equals

Value: Watson Assistant-0s

Add a condition +

Watson Assistant-0s  
Watson Assistant-2j  
Watson Assistant-2n

Next

6. In **Roles and actions**, select the [service role](#) that you want the user to have. Service access controls what a person can do in Watson Assistant. Next, select the [platform role](#) that you want the user to have. Platform access controls a person's ability to access a service instance in IBM Cloud. Then, click **Review**.

**Roles and actions**

What levels of access do you want to assign?

**Service access** ⓘ  
 Reader (13)  
 Writer (17)  
 Manager (19)  
 Logs Reader (1)  
 Version Maker (1)

**Platform access** ⓘ  
 Viewer (13)  
 Operator (22)  
 Editor (30)  
 Administrator (46)

**Manager**

As a manager, you have permissions beyond the writer role to complete privileged actions as defined by the service. In addition, you can create and edit service-specific resources.

**Actions (19)**

GET /conversation  
Can use API endpoints to extract data from skills and assistants

POST /conversation  
Can use API endpoints to create data & to use the message endpoint

DELETE /conversation  
Can use API endpoints to delete data from skills and assistant

Review

7. Click **Add** to add the access policy.

**Platform and service access**

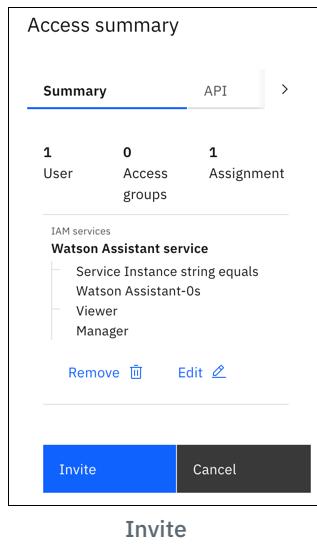
Service  
Watson Assistant

Resources  
Service Instance string equals Watson Assistant-0s

Roles and actions  
Viewer, Manager

Add >

8. To finish, click **Invite**.



Invite

The user that you invited appears in your list with the status of **Processing**. After they accept the invite, status changes to **Active**, and the user can work on your assistant with you.

## Platform roles

A platform role controls a user's ability to open and work with a service instance in IBM Cloud.

**⚠️ Important:** At a minimum, each user needs the *Viewer* platform role for a service instance

Role	Open	Modify	Delete	Manage access
Viewer	<input type="text"/>			
Operator	<input type="text"/>	<input type="text"/>		
Editor	<input type="text"/>	<input type="text"/>		
Administrator	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Platform role details

## Service roles

A service role controls what a person can do within each service instance.

Role	Description
Reader	Read-only access to a resource. Use with Logs Reader to provide access to Analytics.
Writer	Create and edit within a resource.
Manager	Manage everything in a resource.
Logs Reader	Use Logs Reader in combination with the Reader or Writer role to provide access to Analytics.

<b>Version Maker</b>	Create or delete versions of an assistant. Doesn't provide publish access.			
<b>Service role details</b>				
This table explains the minimum service roles that are required for common tasks in an assistant.				
Task	Resource	Minimum service role required		
<b>Assistant</b>				
Create assistant	Service instance	Writer		
View assistant settings	Assistant	Writer		
View assistant ID	Assistant	Writer		
Update assistant settings	Assistant	Writer		
Enable or disable dialog	Assistant	Writer		
Delete assistant	Service instance	Writer		
<b>Actions</b>				
Create action	Action skill	Writer		
Update action	Action skill	Writer		
Delete action	Action skill	Writer		
Download actions JSON file	Action skill	Reader		
Upload actions JSON file	Action skill	Writer		
Copy action	Action skill (in destination assistant)	Writer		
<b>Publish</b>				
Publish version	Environment	Writer		
Create version without publishing	Assistant	Writer or Version Maker		
Delete unpublished version	Assistant	Writer or Version Maker		
Download version	Assistant	Reader		
<b>Environments</b>				
Create environment (Enterprise plan only)	Assistant	Writer		
Update environment settings	Environment	Writer		

Delete environment (Enterprise plan only)	Service instance	Writer
<b>Integrations</b>		
Add integration	Service instance	Writer
Update integration	Service instance	Writer
Delete integration	Service instance	Writer
<b>Dialog</b>		
Create intent	Dialog skill	Writer
Update intent	Dialog skill	Writer
Delete intent	Dialog skill	Writer
Import intents	Dialog skill	Writer
Export intents	Dialog skill	Reader
Create entity	Dialog skill	Writer
Update entity	Dialog skill	Writer
Delete entity	Dialog skill	Writer
Download intents and entities	Dialog skill	Reader

#### Minimum service role details

### Example of limiting access to one assistant

This example explains how to follow the steps in [Adding users from the Manage menu](#) and set specific resources that limit a user to building and publishing actions in one assistant. For each user, you need to add three access policies that identify the service instance, assistant ID, and skill ID. (Use **Assistant settings** to get the ID values for your assistant and action skill.)

This table lists the values that you need to add for each policy:

Policy	Specific resources	Value	Service role	Platform role
1	Service Instance	Choose the instance that includes the assistant	None	Viewer
2	Resource Type	Assistant ID	Manager	None
2	Assistant, Environment, or Skill ID	ID for the assistant	Manager	None
3	Resource Type	Skill ID	Manager	None

3	Assistant, Environment, or Skill ID	ID for the action skill	Manager	None
---	-------------------------------------	-------------------------	---------	------

### Settings to limit access to one assistant

The access policies for your user should look like this example:

Access policies				
Based on your assigned role, you can click the role to view or edit the policy.				
Service	Resources	Role	Conditions	Last permit
Watson Assistant	Resource Type string equals skill, Assistant, Environment or Skill ID string equals 18e19ed0-d4fb-499d-9cc0-163f5db627b4	Manager	2023-02-24	⋮
Watson Assistant	Resource Type string equals assistant, Assistant, Environment or Skill ID string equals f71bbb59-076d-4e17-b9e0-31a6334f4785	Manager	2023-02-24	⋮
Watson Assistant	Service Instance string equals Watson Assistant-0s	Viewer	2023-02-24	⋮
Items per page:	100	1–3 of 3 items	1	1 of 1 page ⏪ ⏩

Access policies example

With this set of access policies, your user can build and publish actions in one assistant. The user can't add integrations because the service instance is set to `Reader`. The user has read-only access to other assistants but can't build or publish actions.

## Managing your plan

This topic provides:

- A [plan information](#) reference
- Steps on [upgrading your plan](#)

### Plan information

Billing for the use of Watson Assistant is managed through your IBM Cloud® account.

The metrics that are used for billing purposes differ based on your plan type. You can be billed based on the number of API calls made to a service instance or on the number of active users who interact with the instance.

For answers to common questions about subscriptions, see [How you're charged](#).

Explore the Watson Assistant [service plan options](#).

### Paid plan features

The following features are available only to users of a Plus or Enterprise plan. 

- [Phone integration](#)
- [Private endpoints](#)
- [Search](#)
- [v2 Logs API](#)
- [Log webhook](#)
- [Autolearning](#)
- [Intent conflict resolution](#)
- [Intent recommendations and intent user example recommendations](#)

The following features are available only to users of Enterprise plans. 

- [Activity tracker](#)

The plan type of the service instance you are currently using is displayed in the page header. You can upgrade from one plan type to another. For more information, see [Upgrading](#).

## User-based plans explained

Unlike API-based plans, which measure usage by the number of API calls made during a month, the Plus and Enterprise plans measure usage by the number of monthly active users.

A *monthly active user (MAU)* is any unique user who has at least one meaningful interaction with your assistant or custom application over the calendar billing month. A meaningful interaction is an exchange in which a user sends a request to your service and your service responds. Welcome messages that are displayed at the start of a conversation are not charged.

A unique user is recognized by the user ID that is associated with the person that interacts with your assistant. The web chat and other built-in integrations set this property for you automatically.

## Specifying the user ID with the REST API

If you are using a custom client with the Watson Assistant API, you must set the `user_id` property in the message payload your client sends to the `message` method. The `user_id` property is specified at the root of the request body, as in this example:

```
$ {
  "input": {
    "message_type": "text",
    "text": "I want to cancel my order"
  },
  "user_id": "my_user_id"
}
```

**⚠ Important:** In some older SDK versions, the `user_id` property is not supported as a top-level method parameter. As an alternative, you can specify `user_id` within the nested `context.global.system` object.

For more information about the `user_id` property, see the API reference documentation:

- [v2 stateless /message](#)
- [v2 stateful /message](#)

## If the user ID is not specified

If you are using a custom client application and do not set a `user_id` value, the service automatically sets it to one of the following values:

- **session\_id (v2 API only):** A property defined in the v2 API that identifies a single conversation between a user and the assistant. A session ID is provided in /message API calls that are generated by the built-in integrations. The session ends when a user closes the chat window or after the inactivity time limit is reached.

**⚠ Note:** If you use the stateless v2 message API, you must specify the `session_id` with each message in an ongoing conversation (in `context.global.session_id`).

- **conversation\_id (v1 API only):** A property defined in the v1 API that is stored in the `context` object of a /message API call. This property can be used to identify multiple /message API calls that are associated with a single conversational exchange with one user. However, the same ID is only used if you explicitly retain the ID and pass it back with each request that is made as part of the same conversation. Otherwise, a new ID is generated for each new /message API call.

For example, if the same person chats with your assistant on three separate occasions over the same billing period, how you represent that user in the API call impacts how the interactions are billed. If you identify the user interaction with a `user_id`, it counts as one use. If you identify the user interaction with a `session_id`, then it counts as three uses (because there is a separate session that is created for each interaction).

Design any custom applications to capture a unique `user_id` or `session_id` and pass the information to Watson Assistant. Choose a non-human-identifiable ID that doesn't change throughout the customer lifecycle. For example, don't use a person's email address as the user ID. In fact, the `user_id` syntax must meet the requirements for header fields as defined in [RFC 7230](#).

The built-in integrations derive the user ID in the following ways:

- For Facebook integrations, the `user_id` property is set to the sender ID that Facebook provides in its payload.
- For Slack integrations, the `user_id` property is a concatenation of the team ID, such as T09LVDR7Y, and the member ID of the user, such as W4F8K9JNF. For example: T09LVDR7YW4F8K9JNF.
- For web chat, you can set the value of the `user_id` property.

Billing is managed per monthly active user per service instance. If a single user interacts with assistants that are hosted by different service instances that belong to the same plan, each interaction is treated as a separate use. You are billed for the user's interaction with each service instance separately.

## Test activity charges

Test messages that you send from the *Preview* button are charged. For the preview, a random `user_id` is generated and stored in a cookie. The multiple interactions that a single tester has with the assistant embedded in the preview are recognized as coming from a single user and are charged accordingly. If you are doing your own test, running a scripted regression test for example, use a single `user_id` for all of the calls within your regression test. Other uses are flagged as abuse.

## Handling anonymous users

If your custom application or assistant interacts with users who are anonymous, you can generate a randomized universally unique ID to represent each anonymous user. For more information about UUIDs, see [RFC 4122](#).

- For web chat, if you do not pass an identifier for the user when the session begins, the web chat creates one for you. It creates a first-party cookie with a generated anonymous ID. The cookie remains active for 45 days. If the same user returns to your site later in the month and chats with your assistant again, the web chat integration recognizes the user. And you are charged only once when the same anonymous user interacts with your assistant multiple times in a single month.

If an anonymous user logs in and later is identified as being the same person who submitted a request with a known ID, you are charged twice. Each message with a unique user ID is charged as an independent active user. To avoid this situation, you can prompt users to log in before you initiate a chat or you can use the anonymous user ID to represent the user consistently.

## Data centers

IBM Cloud has a network of global data centers that provide performance benefits to its cloud services. See [IBM Cloud global data centers](#) for more details.

You can create Watson Assistant service instances that are hosted in the following data center locations:

Location	Location code	API location
Dallas	us-south	N/A
Frankfurt	eu-de	fra

Seoul	kr-seo	seo
Sydney	au-syd	syd
Tokyo	jp-tok	tok
London	eu-gb	lon
Washington DC	us-east	wdc

#### Data center locations

## Upgrading your plan

You can explore the Watson Assistant [service plan options](#) to decide which plan is best for you.

The page header shows the plan you are using today. To upgrade your plan, complete these steps:

1. Do one of the following things:

- **Trial plan only:** The number of days that are left in your trial is displayed in the page header. To upgrade your plan, click **Upgrade** from the page header before your trial period ends.
- For all other plan types, click **Manage** , and then choose **Upgrade** from the menu.

2. From here, you can see other available plan options. For most plan types, you can step through the upgrade process yourself.

- If you upgrade to an Enterprise with Data Isolation plan, you cannot do an in-place upgrade of your service instance. An Enterprise with Data Isolation plan instance must be provisioned for you first.
- When you upgrade from a legacy Standard plan, you change the metrics that are used for billing purposes. Instead of basing billing on API usage, the Plus plan bases billing on the number of monthly active users. If you built a custom app to deploy your assistant, you might need to update the app. Ensure that the API calls from the app include user ID information. For more information, see [User-based plans explained](#).
- You cannot change from a Trial plan to a Lite plan.

For answers to common questions about subscriptions, see the [How you're charged](#).

## Using IBM Cloud Activity Tracker to audit user activity Enterprise

 **IBM Cloud only**

You can use the IBM Cloud Activity Tracker service to track how users and applications interact with IBM Watson® Assistant in IBM Cloud®. This applies to both the classic and new experiences of Watson Assistant.

IBM Cloud Activity Tracker records user-initiated activities that change the state of a service in IBM Cloud. You can use this service to investigate abnormal activity and critical actions and to comply with regulatory audit requirements. In addition, you can be alerted about actions as they happen. The events that are collected comply with the Cloud Auditing Data Federation (CADF) standard. For more information, see [Getting started with IBM Cloud Activity Tracker](#).

## Viewing events

Events that are generated by an instance of the Watson Assistant service are automatically forwarded to the IBM Cloud Activity

Tracker service instance that is available in the same location. However, if your service instance is hosted in the **Washington DC** location, create the IBM Cloud Activity Tracker service instance in the **Dallas** region.

IBM Cloud Activity Tracker can have only one instance per location. To view events, you must access the web UI of the IBM Cloud Activity Tracker service in the same location where your service instance is available. For more information, see [Navigating to the UI](#).

## List of events

The following table lists the Watson Assistant activities that generate events.

**Note:** In IBM Cloud Activity Tracker, each event has the prefix `conversation.`, for example: `conversation.action.create`

Activity	When someone...
<code>action.create</code>	creates an action
<code>action.delete</code>	deletes an action
<code>action.update</code>	updates an action
<code>action_handler.create</code>	creates an action handler
<code>action_handler.delete</code>	deletes an action handler
<code>action_handler.update</code>	updates an action handler
<code>action_variable.create</code>	creates an action variable
<code>action_variable.delete</code>	deletes an action variable
<code>action_variable.update</code>	updates an action variable
<code>actions.copy</code>	copies an action from one assistant to another
<code>assistant.create</code>	creates an assistant

assistant.delete	deletes an assistant
assistant.update	updates an assistant, for example, updates the settings
catalog_integration.create	creates a custom extension
catalog_integration.delete	deletes a custom extension
catalog_integration.update	updates a custom extension
counterexample.create	marks test user input in the "Try it out" pane as being irrelevant or corrects the categorization of a user input that was incorrectly assigned to an intent by marking it as irrelevant
counterexample.delete	deletes a counterexample
counterexample.update	edits a counterexample
data.delete	deletes multiple training data items, such as multiple entities or intents
data.update	does a bulk action, such as importing a CSV file of intents or entities to the skill
data_type.create	creates a saved response
data_type.delete	deletes a saved response
data_type.update	updates a saved response
entity.create	creates an entity
entity.delete	deletes an entity
entity.update	edits an entity
environment.create	adds an environment
environment.delete	deletes an environment
environment.update	updates an environment
example.create	adds a user input example to an intent
example.delete	deletes a user example from an intent
example.update	edits a user example that is associated with an intent
integration_defintion.create	creates an integration
integration_defintion.delete	deletes an integration

integration_definition.update	updates an integration
intent.create	creates an intent
intent.delete	deletes an intent
intent.update	edits an intent
log.create	corrects an intent that was inaccurately categorized by the skill from the Analytics>User conversations page
node.create	creates a dialog node
node.delete	deletes a dialog node
node.update	edits a dialog node
notifier.create	creates a notifier
notifier.delete	deletes a notifier
notifier.update	updates a notifier
processor.create	creates a processor
processor.delete	deletes a processor
processor.update	updates a processor
recommendationfile.create	uploads a CSV file of utterances to a skill from which Watson can derive intent recommendations
recommendationfile.delete	deletes a CSV file of utterances that is used to derive intent recommendations from a skill.
recommendationsources.update	updates a CSV file or assistant log that is being used as the source for intent recommendations
release.create	create a version from content in the draft environment
release.delete	delete a version
release.deploy	publish a version to an environment
skill.create	creates a skill
skill.delete	deletes a skill
skill.update	updates a skill

skill_reference.create	adds a specific skill to an assistant
skill_reference.delete	removes a specific skill from an assistant
skill_reference.update	updates a specific skill that is associated with an assistant
skill_variable.create	create a skill variable
skill_variable.delete	delete a skill variable
skill_variable.update	update a skill variable
skills.export	export a skill
skills.import	import a skill
snapshot.create	creates a version of a dialog skill
snapshot.delete	deletes a version of a dialog skill
snapshot.update	updates a version of a dialog skill
step.create	adds a step to an action
step.delete	deletes a step from an action
step.update	updates a step in an action
step_handler.create	create a step handler
step_handler.delete	delete a step handler
step_handler.update	update a step handler
synonym.create	creates a synonym for an entity value
synonym.delete	deletes a synonym that is associated with an entity value
synonym.update	edits a synonym that is associated with an entity value
userdata.delete	deletes data that was created by a specified customer
value.create	creates an entity value
value.delete	deletes an entity value
value.update	edits an entity value

<code>workspace.create</code>	creates a workspace
<code>workspace.delete</code>	deletes a workspace
<code>workspace.update</code>	makes changes to a workspace

Table 1. Activity that generates events

## Additional information for update events

Any of the update events include this additional information in a `requestData.update` object:

- Name changed
- Title changed
- Metadata changed
- Training data changed

This example from an `assistant.update` event shows a name change:

```
$ "update": [{"updateType": "Name changed", "nameAttribute": "name", "newValue": "Banking Bot 2"}, {"updateType": "Metadata changed", "attributesUpdated": ["description", "language"]}], "environment": "draft"},
```

## Securing your assistant

IBM is committed to providing our clients and partners with innovative data privacy, security, and governance solutions.

**Notice:** Clients are responsible for ensuring their own compliance with various laws and regulations, including the European Union General Data Protection Regulation (GDPR). Clients are solely responsible for obtaining advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulations that may affect the clients' business and any actions the clients may need to take to comply with such laws and regulations.

The products, services, and other capabilities described herein are not suitable for all client situations and may have restricted availability. IBM does not provide legal, accounting or auditing advice or represent or warrant that its services or products will ensure that clients are in compliance with any law or regulation.

If you need to request GDPR support for IBM Cloud® Watson resources that are created

- In the European Union, see [Requesting support for IBM Cloud Watson resources created in the European Union](#)
- Outside the European Union, see [Requesting support for resources outside the European Union](#).

## European Union General Data Protection Regulation (GDPR)

IBM is committed to providing our clients and partners with innovative data privacy, security and governance solutions to assist them on their journey to GDPR compliance.

Learn more about IBM's own GDPR readiness journey and our GDPR capabilities and offerings to support your compliance journey [here](#).

## Health Insurance Portability and Accountability Act (HIPAA)

US Health Insurance Portability and Accountability Act (HIPAA) support is available for Enterprise plans that are hosted in the Washington, DC or Dallas locations. For more information, see [Enabling HIPAA support for your account](#).

Do not add personal health information (PHI) to the training data (entities and intents, including user examples) that you create. In particular, be sure to remove any PHI from files that contain real user utterances that you upload to mine for intent or intent user example recommendations.

## Opting out of log data use

IBM uses log data, Enterprise plan data excluded, to continually learn from and improve the Watson Assistant product. The logged data is not shared or made public.

To prevent IBM from using your log data for general service improvements, complete one of the following tasks:

- If you are using a custom application, for each API `/message` request, set the `X-Watson-Learning-Opt-Out` header parameter to `true`.

For more information, see [Data collection](#).

- If you are using the web chat integration, add the `learningOptOut` parameter to the script that you embed in your web page, and set it to `true`.

For more information, see [Configuration](#).

## Labeling and deleting data in Watson Assistant

Do not add personal data to the training data (actions and steps, including user examples) that you create. In particular, be sure to remove any personally-identifiable information from files that contain real user utterances that you upload to mine for user example recommendations.

Experimental and beta features are not intended for use with a production environment and therefore are not guaranteed to function as expected when labeling and deleting data. Experimental and beta features should not be used when implementing a solution that requires the labeling and deletion of data.

If you need to remove a customer's message data from a Watson Assistant instance, you can do so based on the customer ID of the client, as long as you associate the message with a customer ID when the message is sent to Watson Assistant.

 **Note:** Removing message data must be an occasional event only for individual customer IDs. To disable analytics logs, you can upgrade to an Enterprise with Data Isolation plan.

- The assistant preview and automatic Facebook integration do not support the labeling and therefore deletion of data based on customer ID. They should not be used in a solution that must support the ability to delete data based on a customer ID.
- For Intercom, the `customer_id` is the `user_id` prepended with `intercom_`. The Intercom `user_id` property is the `id` of the `author` message object in the Conversation Model that is defined by Intercom.
  - To get the ID, open the channel from a web browser. Open the web developer tools to view the console. Look for `author`.

The full customer ID looks like this: `customer_id=intercom_5c499e5535ddf5c7fa2d72b3`.

- For Slack, the `customer_id` is the `user_id` prepended with `slack_`. The Slack `user_id` property is a concatenation of the team ID, such as `T09LVDR7Y`, and the member ID of the user, such as `W4F8K9JNF`. For example: `T09LVDR7YW4F8K9JNF`.

- To get the team ID, open the channel from a web browser. Open the web developer tools to view the console. Look

- for [BOOT] Initial team ID.
- o You can copy the member ID from the user's Slack profile.
- o To get the IDs programmatically, use the Slack API. For more information, see [Overview](#). The full customer ID looks like this: `customer_id=slack_T09LVDR7YW4F8K9JNF`.
- For the web chat integration, the service takes the `user_id` that is passed in and adds it as the `customer_id` parameter value to the `X-Watson-Metadata` header with each request.

## Before you begin

To be able to delete message data associated with a specific user, you must first associate all messages with a unique **customer ID** for each user. To specify the **customer ID** for any messages sent using the `/message` API, include the `X-Watson-Metadata: customer_id` property in your header. For example:

```
$ curl -X POST -u "apikey:3Df... ...Y7Pc9"
--header
'Content-Type: application/json'
'X-Watson-Metadata: customer_id=abc'
--data
'{"input":{"text":"hello"}}'
'{url}/v2/assistants/{assistant_id}/sessions/{session_id}/message?version=2019-02-28'
```

where `{url}` is the appropriate URL for your instance. For more details, see [Service endpoint](#).

 **Note:** The `customer_id` string cannot include the semicolon (;) or equal sign (=) characters. You are responsible for ensuring that each `customer ID` property is unique across your customers.

Only the first **customer ID** value that is passed in the `X-Watson-Metadata` header is used as the `customer_id` string for the message log. This **customer ID** value can be deleted with `DELETE /user_data` v1 API calls.

If you add search to an assistant, user input that is submitted to the assistant is passed to the Discovery service as a search query. If the Watson Assistant integration provides a customer ID, then the resulting `/message` API request includes the customer ID in the header, and the ID is passed through to the Discovery `/query` API request. To delete any query data that is associated with a specific customer, you must send a separate delete request directly to the Discovery service instance that is linked to your assistant. See the Discovery [information security](#) topic for details.

## Querying user data

Use the v1 `/logs` method `filter` parameter to search an application log for specific user data. For example, to search for data specific to a `customer_id` that matches `my_best_customer`, the query might be:

```
$ curl -X GET -u "apikey:3Df... ...Y7Pc9" \
'{url}/v2/assistants/{assistant_id}/logs?version=2020-04-01&filter=customer_id::my_best_customer"
```

where `{url}` is the appropriate URL for your instance. For more details, see [Service endpoint](#).

See the [Filter query reference](#) for additional details.

## Deleting data

To delete any message log data associated with a specific user that your assistant might have stored, use the `DELETE /user_data` v1 API method. Specify the customer ID of the user by passing a `customer_id` parameter with the request.

Only data that was added by using the `POST /message` API endpoint with an associated customer ID can be deleted using this delete method. Data that was added by other methods cannot be deleted based on customer ID. For example, entities and

intents that were added from customer conversations, cannot be deleted in this way. Personal Data is not supported for those methods.

**IMPORTANT:** Specifying a `customer_id` will delete *all* messages with that `customer_id` that were received before the delete request, across your entire Watson Assistant instance, not just within one skill.

As an example, to delete any message data associated with a user that has the customer ID `abc` from your Watson Assistant instance, send the following cURL command:

```
$ curl -X DELETE -u "apikey:3Df... ...Y7Pc9" \
"{{url}}/v2/user_data?customer_id=abc&version=2020-04-01"
```

where `{url}` is the appropriate URL for your instance. For more details, see [Service endpoint](#).

An empty JSON object `{}` is returned.

For more information, see the [API reference](#).

**Note:** Delete requests are processed in batches and may take up to 24 hours to complete.

## Web chat usage data

The Watson Assistant web chat sends limited usage data to the [Amplitude service](#). When the web chat widget is being interacted with by a user, we track the features that are being used, and events such as how many times the widget is opened and how many users start conversations. This information does not include Assistant training data or the content of any chat interactions. The information being sent to Amplitude is not Content as defined in the Cloud Service Agreement (CSA); it is Account Usage Information as described in Section 9.d of the CSA and is handled accordingly as described in the [IBM Privacy Statement](#). The purpose of this information gathering is limited to establishing statistics about use and effectiveness of the web chat and making general improvements.

## Private network endpoints

You can set up a private network for Watson Assistant instances that are part of a Plus or Enterprise service plan. Using a private network prevents data from being transferred over the public internet, and ensures greater data isolation.

 This feature is available only to users of paid plans.

Private network endpoints support routing services over the IBM Cloud private network instead of the public network. A private network endpoint provides a unique IP address that is accessible to you without a VPN connection.

For implementation details, see [Public and private network endpoints](#).

## Important private network endpoint notes

- The integrations that are provided with the product require endpoints that are available on the public internet. Therefore, any built-in integrations you add to your assistant will have public endpoints. If you only want to connect to a client application or messaging channel over the private network, then you must build your own custom client application or channel integration.
- Before you can use a search integration or search skill, you must create a Discovery instance with a private network endpoint. The list of Discovery instances that are displayed for you to connect to includes only instances with private network endpoints.

## Related topics

- [Security architecture](#): Describes the security components that are needed for secure cloud development, deployment, and operations.

- [IBM Cloud compliance programs](#): Describes how to manage regulatory compliance and internal governance requirements with IBM Cloud services.

## Backing up and restoring data

---

Back up and restore your Watson Assistant data by downloading, and then uploading the data.

You can download the following data from a Watson Assistant service instance:

- Actions

You cannot download the following data:

- Search
- Assistant, including any configured integrations

## Retaining logs

If you want to store logs of conversations that users have had with your assistant, you can use the `/logs` API to export your log data. See [API reference](#) for details.

Logs are stored for a different amount of time depending on your service plan. For example, Lite plans provide logs from the past 7 days only. See [Log limits](#) for more information.

## Downloading

To back up actions, download a JSON file and store it.

1. On the **Actions** page, click **Global settings** .
2. On the **Upload/Download** tab, click the **Download** button.

## Uploading

To reinstate a backup copy of actions that you exported from another service instance or environment, import the JSON file of the actions you exported.

 **Important:** If the Watson Assistant service changes between the time you export the actions and import it, due to functional updates that are regularly applied to instances in cloud-hosted continuous delivery environments, your imported actions might function differently than before.

1. On the **Actions** page, click **Global settings** .
2. On the **Upload/Download** tab, drag and drop a JSON file onto the tab or click to select a file from your local system, then click **Upload**.

 **Important:** The imported JSON file must use UTF-8 encoding, without byte order mark (BOM) encoding. The JSON file cannot contain tabs, newlines, or carriage returns.

## High availability and disaster recovery

---

IBM Watson® Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.

You are responsible for understanding your configuration, customization, and usage of Watson Assistant. You are also responsible for being ready to re-create an instance of the service in a new location and to restore your data in any location. See [How do I ensure zero downtime?](#) for more information.

## High availability

Watson Assistant supports high availability with no single point of failure. The service achieves high availability automatically and transparently by using the multi-zone region feature provided by IBM Cloud.

IBM Cloud enables multiple zones that do not share a single point of failure within a single location. It also provides automatic load balancing across the zones within a region.

## Disaster recovery

Disaster recovery can become an issue if an IBM Cloud location experiences a significant failure that includes the potential loss of data. Because the multi-zone region feature is not available across locations, you must wait for IBM to bring a location back online if it becomes unavailable. If underlying data services are compromised by the failure, you must also wait for IBM to restore those data services.

If a catastrophic failure occurs, IBM might not be able to recover data from database backups. In this case, you need to restore your data to return your service instance to its most recent state. You can restore the data to the same or to a different location.

Your disaster recovery plan includes knowing, preserving, and being prepared to restore all data that is maintained on IBM Cloud. See [Backing up and restoring data](#) for information about how to back up your service instances.

## Failover options

---

This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.

### Introduction

IBM Watson® Assistant instances are deployed across multizone regions (MZRs) in all data centers (except Seoul, Korea) and can withstand the loss of any individual zone within an MZR. However, regional outages can occur and while IBM strives to keep these to a minimum, you may want to consider moving traffic to a different region for your business-critical applications.

To achieve this, you should have a copy of your Watson Assistant instance in a different MZR (for example, have your assistant deployed in US East and US South). To do this, you must purchase a second service instance of Watson Assistant, and second instances of the necessary assistants and skills need to be instantiated. You must implement change management controls to synchronize the changes between the two regions. Watson Assistant and Speech services have APIs available to export and import the definitions.

With two instances, there are two topologies to consider:

- **Active/active:** Both instances always serving traffic with the load sprayed between the two.
- **Active/passive:** One instance is active and the passive site only receives traffic in the event of a failover.

There are pros and cons to both approaches, and considerations specific to Watson Assistant are detailed in the sections that follow.

### Considerations

Watson Assistant instances in one region are unaware of instances in a second region, which can affect some features and capabilities in Watson Assistant.

## Analytics

Watson Assistant analytics provide overview statistics on the number of interactions with users and containment rates. Analytics doesn't cumulate statistics across regions. With an active/passive topology, this approach to analytics should be sufficient. However, using an active/active topology likely requires using [webhooks](#) to gather interaction data, and build custom data warehouses and reports to understand total usage.

## Session history for web chat and the v2 api

Session history allows your web chats to maintain conversation history and context when users refresh a page or change to a different page on the same website. This feature doesn't work across instances, so in-progress conversations need to be restarted.

## Billing

IBM calculates your bill based on the IBM Cloud Account. Watson Assistant calculates monthly average user (MAU) metrics by aggregating within a given service instance as follows:

- The same MAU used in 2 different assistant resources in the **same** service instance counts as 1 MAU
- The same MAU used in 2 different assistant resources in **different** service instances counts as 2 MAUs

Note that for an **active/active** topology, under the worst case scenario, the MAU count could end up being doubled for a given billing period.

## Phone integration

A Watson Assistant phone integration in one region is unaware of a phone integration in a different region. You need to ensure that your assistants are identically configured in both regions. You also need to rely on the upstream SIP trunking provider to detect and manage failover between regions.

## Monitoring

SIP trunking providers can be configured to actively health-check the Watson Assistant session border controllers (SBCs) by sending periodic SIP OPTIONS messages to each zone within a region. A failure to receive a response from one of the zones being health-checked can be used to either provide notification of a failure to trigger a manual failover, or it can be used to automate removal of the failed zone from the route list.

## Failover

The SIP trunking provider plays an important role in detecting and managing a failover, especially if an automatic failover is expected between regions. In most cases, SIP trunking providers should be configured to treat each zone within a region as active/active and two regions where an assistant is configured as active/passive. SIP trunking providers should always be configured to load balance and fail over between zones within a single region. The remainder of this section focuses on handling a failover between regions.

## Full service outage

Phone integration failures have two types. The first type is a full outage where the session border controllers in all 3 regional zones become unreachable. This is the easier of the two types to detect and handle because the SIP trunking provider is immediately notified via SIP timeouts that the call fails and can be configured to either automatically fail over or the call routing can be manually reconfigured at the SIP trunking provider to direct traffic away from the failed region towards the passive backup region. If a failover is automated and a regional backup is enabled, it is always best to try a different zone first and only redirect traffic to the passive backup region if a preconfigured number of failures occur within a short period of time. This prevents an unnecessary failover between regions if only a short outage occurs.

Note that Watson Assistant provides a round-robin fully qualified domain name (FQDN) that includes the IPs for each zone in the region. Many SIP trunking providers automatically retry each IP in the FQDN when failures occur. To support disaster recovery, the service provider may need to configure two separate SIP trunks, one for each region, and only when all the zones in a single region fail should the call be switched to the backup region. It's important to set the SIP INVITE failure timeouts at the SIP trunking provider low enough to avoid long call setup latencies when a failover is occurring.

## Partial service outage

The second type of failure is a partial service outage within the region. This is much harder to detect and manage because of the large number of variations in service failures that can occur within a region. In some cases, these may be small issues that affect the performance characteristics of the call but not cause the call to fail.

For issues like this that ultimately cause a call to fail, there are two ways Watson Assistant can handle the call. The first is to accept the call and then transfer it to a configured default SIP URI. This can be configured in the Watson Assistant phone integration settings and is also used if there is a mid-call failure. The default transfer target SIP URI is defined in the **SIP target when a call fails** field located on the Advanced tab of the phone integration configuration panel.

The phone integration can also be configured to respond to a SIP INVITE with a SIP 500 (service unavailable) message if an outage is detected during call setup instead of transferring a call to a live agent. A SIP 500 can then be used to redirect the call to another zone, or if many SIP 500s are received, to another region. Using a SIP 500 INVITE error is a better way to signal a failure to an upstream SIP trunking provider because it gives the provider a way to reroute the call. Using only the default transfer target to handle call failures is acceptable for low call volume scenarios but can result in large numbers of calls being directed to a customer contact center when bigger call volumes are handled by Watson Assistant. To enable this 500 error response capability for a specific Watson Assistant instance, you need to make a request to IBM.

You should plan for both full and partial service outages. A good first step is to plan for a manual failover between regions before enabling automation. This requires a complete replica of Watson Assistant in both regions, including all custom speech model training. When automation is enabled it is best to start with a strategy for detecting and failing over to the passive backup region when a complete regional outage is detected. After this is in place, a strategy can be implemented to deal with partial outages, which should cover the vast majority of failure conditions that can occur with a phone integration deployment.

## Web chat

### Monitoring

Web chat provides an `onError` listening feature that allows the host page to detect specific types of outage errors, in particular INITIAL\_CONFIG, OPEN\_CONFIG, and MESSAGE\_COMMUNICATION errors.

You can find the documentation for this feature here: <https://web-chat.global.assistant.watson.cloud.ibm.com/docs.html?to=api-configuration#onerror-detail>

### Failover

Handling a failover for web chat is simple assuming you have set up an additional web chat integration in another region. When the failover needs to be manually triggered, make the following changes:

- The embed script that contains your integration ID, region, service instance ID, and subscription ID (if applicable) needs to be changed or updated to use the IDs for the new integration and region.
- If you are using Salesforce or ZenDesk integrations for connecting to human agents, update the configuration within those systems to make sure they can communicate with the correct integration. Follow the instructions on the **Live agent** tab in the web chat configuration for setting up those systems. This is only needed for obtaining the conversation history for the agent.
- If you have web chat security enabled and you are using encrypted payloads, the IBM-provided public key used for the

encryption may be different depending on region. If so, you need to update the system that generates the JSON Web Token (JWT) to use the correct key.

You can set up an active/active configuration of web chat by using different integration IDs in your embed scripts, and by ensuring the web chat integration is "sticky" by user. Otherwise, if the user fails over to a different integration, the conversation history might be lost.

## API

### Monitoring

Capture and monitor response codes of /message calls. Response codes of live /message traffic should be captured and aggregated.

Ideally, save the response code statistics in an external persistence store. This shared store can aggregate response code results from all deployed instances of a your application and provide the greatest fidelity in determining if a failover should be triggered.

Alternatively, response codes can be aggregated and evaluated in memory for a given instance of your application. As only a fraction of traffic is contributing to the failover decision, this could lead to different behavior with respect to how often the failover decision is acted upon.

With either aggregation approach, exceeding a defined threshold could trigger a failover. Common approaches for determining a failover include:

- Percentage-based: greater than X% of requests return a non-200 response code
- Consecutive-based: X calls in a row return a non-200 response code
- Limit-based: X calls return a non-200 response in a given timeframe

To avoid an unnecessary failover between regions, make sure a robust retry mechanism is present when calling the /message endpoint.

### Failover for v1 and v2 stateless API

For a successful failover, ensure the following:

- Training data changes should be synchronized across regions. Avoid pushing changes delayed over a large window of time (such as days) to mitigate risk of algorithm changes being deployed by Watson Assistant in between regions being updated.
- The same IBM Cloud account should be used for the service instances across regions to maintain a single overall bill for services.
- The client applications should support:
  - Watson Assistant API hostname
  - Service instance credentials
  - v1: workspace\_id
  - v2: assistant\_id

Although it does not affect the runtime flow of calling /message, if you are using fine-grained access control using IBM Access Control (IAM), make sure the IAM policies are appropriately synchronized across the regions.

Note that IAM is a global service, but the custom resources (assistants and skills) used by Watson Assistant access control means each region, which has specific resources, requires specific policies.

For an **active/passive** topology, some form of a [circuit break pattern](#) can be used. A single service instance in a given region is used exclusively unless errors are detected. At that point, the system can respond by updating the relevant failover metadata to route traffic to the service instance in the other region. Once a failover happens, you can decide to continue using the new region

as the active instance, or if you want to resume using the initial region once it has stabilized.

For an **active/active** topology, some form of a load balancing can be used, where two or more service instances in unique regions always receive a percentage of traffic. Additional logic would need to be established to determine when to pull a region out of rotation. This monitoring logic could use a [circuit break pattern](#) similar to the active/passive configuration or rely on a separate dedicated monitoring framework that determines region health. Also similar to active/passive, determining when to insert a region back in rotation would need to be considered as well.

## Failover for v2 stateful API

Failover for the v2 stateful API is similar to stateless, with these details to consider:

- The state of a given conversation is persisted by Watson Assistant in a database that is tied to a particular region. As such, a failover for the stateful v2 /message may more disruptive.
- For an **active/passive** topology, you should assume that all in-progress conversations are ended.
- For an **active/active** topology, given the region-locked persistence constraints of the v2 stateful /message architecture, all turns (/message API calls) of a given conversation (session) should occur within the same region.

## Adding support for global audiences

---

Your customers come from all around the globe. You need an assistant that can talk to them in their own language and in a familiar style. Choose the approach that best fits your business needs.

- **Quickest solution:** The simplest way to add language support is to author the assistant in a single language. You can translate each message that is sent to your assistant from the customer's local language to the assistant language. Later you can translate each response from the assistant language back to the customer's local language.

This approach simplifies the process of authoring and maintaining the conversation. You can build one assistant and use it for all languages. However, the intention and meaning of the customer message can be lost in the translation.

For more information about webhooks you can use for translation, see [Webhook overview](#).

- **Most precise solution:** If you have the time and resources, the best user experience can be achieved when you build multiple assistants, one for each language that you want to support. Watson Assistant has built-in support for all languages. Use one of 13 language-specific models or the universal model, which adapts to any other language you want to support.

When you build an assistant that is dedicated to a language, a language-specific classifier model is used by the assistant. The precision of the model means that your assistant can better understand and recognize the goals of even the most colloquial message from a customer.

Use the universal language model to create an assistant that is fluent even in languages that Watson Assistant doesn't support with built-in models.

To deploy, use the web chat integration with your French-speaking assistant to deploy to a French-language page on your website. Deploy your German-speaking assistant to the German page of your website. Maybe you have a support phone number for French customers. You can configure your French-speaking assistant to answer those calls, and configure another phone number that German customers can use.

## Understanding the universal language model

An assistant that uses the universal language model applies a set of shared linguistic characteristics and rules from multiple languages as a starting point. It then learns from the training data that you add to it.

The universal language classifier can adapt to a single language per assistant. It cannot be used to support multiple languages

within a single assistant. However, you can use the universal language model in one assistant to support one language, such as Russian, and in another assistant to support another language, such as Hindi. The key is to add enough training examples or intent user examples in your target language to teach the model about the unique syntactic and grammatical rules of the language.

Use the universal language model when you want to create a conversation in a language for which no dedicated language model is available, and which is unique enough that an existing model is insufficient.

For more information about feature support in the universal language model, see [Supported languages](#).

## Integration considerations

Keep these tips in mind for integrations:

- **Phone integration:** If you want to deploy an assistant that uses the universal language model with the phone integration, you must connect to custom Speech service language models that can understand the language you're using. For more information about supported language models, see the [Speech to Text](#) and [Text to Speech](#) documentation.
- **Search integration:** If you build an assistant that specializes in a single language, be sure to connect it to data collections that are written in that language. For more information about the languages that are supported by Discovery, see [Language support](#).
- **Web chat:** Web chat has some hardcoded strings that you can customize to reflect your target language. For more information, see [Global audience support](#).

## Supported languages

Watson Assistant supports individual features to varying degrees per language.

Watson Assistant has classifier models that are designed specifically to support conversations in the following languages:

Language	Language code
Arabic	ar
Chinese (Simplified)	zh-cn
Chinese (Traditional)	zh-tw
Czech	cs
Dutch	nl
English	en-us
French	fr
German	de
Italian	it
Japanese	ja
Korean	ko
Portuguese (Brazilian)	pt-br

Spanish	es
Universal	xx

Table 1. Supported languages

## Feature support details

The following tables illustrate the level of language support available for product features.

In the following tables, the level of language and feature support is indicated by these codes:

- GA: The feature is generally available and supported for this language. Note that features might continue to be updated even after they are generally available.
- Beta: The feature is supported only as a Beta release, and is still undergoing testing before it is made generally available in this language.
- NA: Indicates that a feature is not available in this language.

## Content support details

Language	Actions	Search
English (en)	GA	GA
Arabic (ar)	GA	GA
Chinese (Simplified) (zh-cn)	GA	GA
Chinese (Traditional) (zh-tw)	GA	GA
Czech (cs)	GA	GA
Dutch (nl)	GA	GA
French (fr)	GA	GA
German (de)	GA	GA
Italian (it)	GA	GA
Japanese (ja)	GA	GA
Korean (ko)	GA	GA
Portuguese (Brazilian) (pt-br)	GA	GA
Spanish (es)	GA	GA
Universal (xx)	GA	GA

Table 2. Content support details

 **Note:** The Watson Assistant service supports multiple languages as noted, but the tool interface itself (descriptions, labels, etc.) is in English. All supported languages can be input and trained through the English interface.

---

GB18030 compliance: GB18030 is a Chinese standard that specifies an extended code page for use in the Chinese market. This code page standard is important for the software industry because the China National Information Technology Standardization Technical Committee has mandated that any software application that is released for the Chinese market after September 1, 2001, be enabled for GB18030. The Watson Assistant service supports this encoding, and is certified GB18030-compliant

## Changing an assistant language

Once an assistant has been created, its language cannot be modified.

## Working with accented characters

In a conversational setting, users might or might not use accents while interacting with the Watson Assistant service. As such, both accented and non-accented versions of words might be treated the same for intent detection and entity recognition.

However for some languages, like Spanish, some accents can alter the meaning of the entity. Thus, for entity detection, although the original entity might implicitly have an accent, your assistant can also match the non-accented version of the same entity, but with a slightly lower confidence score.

For example, for the word "barrió", which has an accent and corresponds to the past tense of the verb "barrer" (to sweep), your assistant can also match the word "barrio" (neighborhood), but with a slightly lower confidence.

The system will provide the highest confidence scores in entities with exact matches. For example, `barrio` will not be detected if `barrió` is in the training set; and `barrió` will not be detected if `barrio` is in the training set.

You are expected to train the system with the proper characters and accents. For example, if you are expecting `barrió` as a response, then you should put `barrió` into the training set.

Although not an accent mark, the same applies to words using, for example, the Spanish letter `ñ` vs. the letter `n`, such as "uña" vs. "una". In this case the letter `ñ` is not simply an `n` with an accent; it is a unique, Spanish-specific letter.

## Switching between new and classic experiences

---

You can easily switch back and forth between the new experience and the classic experience. However, the new experience provides a simplified user interface, an improved deployment process, and access to the latest features. For more information, see [Migrating to the new experience](#).

To switch between the new and classic experiences, following these steps:

1. From the Watson Assistant interface, click the **Manage**  icon to open your account menu.
2. Select **Switch to classic experience** or **Switch to new experience** from the account menu.

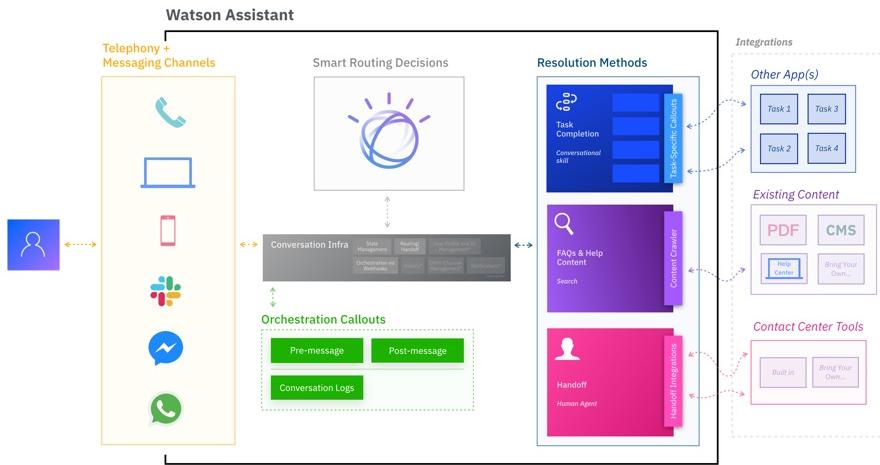
You won't lose any work if you switch to a different experience, and other users of the same instance are not affected. However, keep in mind that any work you do in one experience is not available in the other experience. You can switch back and forth at any time.

# Customizing and developing

## Overview: Customizing and developing

The Watson Assistant user interface makes it easy to build an assistant and deploy it to your customers without writing any code. For advanced users and developers, there are powerful ways you can further customize and extend the capabilities of your assistant.

A deployed assistant includes numerous components that work together to deliver the help your customers need over the channels they use.



- Customers interact with the assistant using a **channel** such as the web chat or phone integration.
- Based on natural language understanding, the assistant makes a decision about how to route the customer's request to the appropriate resolution mechanism, which might be an action or a search of existing content.
- The assistant might also need to communicate with external services or hand off the conversation to a human agent.

There are multiple points at which a developer can customize and extend how the assistant behaves, or how it interacts with external services. These customization points include the following:

- Customizing actions: By writing expressions and editing JSON data, you can extensively customize how an action evaluates step conditions, how it stores data, how it responds to customer input, and how it interacts with channels. (More information coming soon.)
- [Web chat development overview](#): You can use the web chat API to extensively customize the appearance and behavior of the web chat.
- Customizing the phone integration: You can use commands and context variables to extensively configure how your assistant interacts with users using the phone integration. (More information coming soon.)
- Customizing the SMS integration: You can use commands and context variables to customize how your assistant interacts with users using text messages. (More information coming soon.)
- [Extending your assistant using webhooks](#): You use webhooks to call external services that extend the capabilities of your assistant or log activity.
- Developing a custom channel: If none of the built-in channel integrations meet your needs, you can use the Watson Assistant REST API and SDKs to develop a custom client application that interacts with your assistant. (More information coming soon.)

coming soon.)

## Defining responses using the JSON editor

In some situations, you might need to define your assistant's responses using the JSON editor. (For more information about assistant responses, see [Adding assistant responses](#).)

To edit a response using the JSON editor, click the **Switch to JSON editor** icon  in the corner of the **Assistant says** field. The JSON editor shows how the response is defined behind the scenes and sent to the channel.

### Generic JSON format

If you open the JSON editor on a new, empty response, you see the following basic structure:

```
{  
  "generic": []  
}
```

The `generic` property defines an array of responses that will be sent to the channel when the step is executed. The term *generic* refers to the fact that these responses are defined using a generic JSON format that is not specific to any channel. This format can accommodate various response types that are supported by multiple integrations, and can also be implemented by a custom client application that uses the REST API.

The `generic` array for a step can contain multiple responses, and each response has a *response type*. A basic step that sends a simple text response typically includes only a single response with the response type `text`. However, many other response types are available, supporting multimedia and interactive content, as well as control over the behavior of some channel integrations.

Although the `generic` format can be sent to any channel integration, not all channels support all response types, so a particular response might be ignored (or handled differently) by some channels. For information about which channels support which response types, see [Response types](#).

 **Note:** At run time, output containing multiple responses might be split into multiple message payloads. The channel integration sends these messages to the channel in sequence, but it is the responsibility of the channel to deliver these messages to the end user; this can be affected by network or server issues.

### Adding responses

To specify a response in the JSON editor, insert the appropriate JSON objects into the `generic` field of the step response. The following example shows output containing two responses of different types (text and an image):

```
{  
  "generic": [  
    {  
      "response_type": "text",  
      "values": [  
        {  
          "text_expression": {  
            "concat": [  
              {  
                "scalar": "This is a text response."  
              }  
            ]  
          }  
        }  
      ]  
    }  
  ]
```

```

    },
    {
      "response_type": "image",
      "source": "https://example.com/image.jpg",
      "title": "Example image",
      "description": "This is an image response."
    }
  ]
}

```

For information about the available response types, see [Response types](#).

## Targeting specific integrations

If you plan to deploy your assistant to multiple channels, you might want to send different responses to different integrations based on the capabilities of each channel. The `channels` property of the generic response object provides a way to do this.

This mechanism is useful if your conversation flow does not change based on the integration in use, and if you cannot know in advance what integration the response will be sent to at run time. By using `channels`, you can define a single step that supports all integrations, while still customizing the output for each channel. For example, you might want to customize the text formatting, or even send different response types, based on what the channel supports.

Using `channels` is particularly useful in conjunction with the `channel_transfer` response type. Because the message output is processed both by the channel initiating the `transfer` and by the target channel, you can use `channels` to define responses that will only be processed by one or the other. (For more information, and an example, see [Channel transfer](#).)

To specify the integrations for which a response is intended, include the optional `channels` array as part of the response object. All response types support the `channels` array. This array contains one or more objects using the following syntax:

```

{
  "channel": "<channel_name>"
}

```

The value of `<channel_name>` can be any of the following strings:

- `chat`: Web chat
- `voice_telephony`: Phone
- `text.messaging`: SMS with Twilio
- `slack`: Slack
- `facebook`: Facebook Messenger
- `whatsapp`: WhatsApp

The following example shows step output that contains two responses: one intended for the web chat integration and one intended for the Slack and Facebook integrations.

```

{
  "generic": [
    {
      "response_type": "text",
      "channels": [
        {
          "channel": "chat"
        }
      ],
      "values": [
        {
          "text_expression": {
            "concat": [
              {
                "scalar": "This output is intended for the <strong>web chat</strong>."
              }
            ]
          }
        }
      ]
    }
  ]
}

```

```

        }
    ]
}
],
{
  "response_type": "text",
  "channels": [
    {
      "channel": "slack"
    },
    {
      "channel": "facebook"
    }
  ],
  "values": [
    {
      "text_expression": {
        "concat": [
          {
            "scalar": "This output is intended for either Slack or Facebook."
          }
        ]
      }
    }
  ]
}

```

If the `channels` array is present, it must contain at least one channel object. Any integration that is not listed ignores the response. If the `channels` array is absent, all integrations handle the response.

## Response types

The following response types are available. (For detailed information about how to specify each response type, see [Response types reference](#).)

 **Note:** Not all channel integrations support all response types. For information about which integrations support which response types, see [Channel integration support for response types](#).

### audio

Plays an audio clip specified by a URL.

### channel\_transfer

Requests that the conversation be transferred to a different integration. (Currently, only the web chat integration can be the target of a channel transfer.)

### connect\_to\_agent

Requests that the conversation be transferred to a contact center with live agents for help.

### date

Requests that the channel collect a date value from the customer (for example, by displaying an interactive calendar).

### dtmf

Sends commands to the phone integration to control input or output using dual-tone multi-frequency (DTMF) signals. (DTMF is a protocol used to transmit the tones that are generated when a user presses keys on a push-button phone.)

### end\_session

Sends a command to the channel ending the session. This response type instructs the phone integration to hang up the call.

#### iframe

Embeds content from an external website as an HTML `iframe` element.

#### image

Displays an image specified by a URL.

#### option

Presents a set of options (such as buttons or a drop-down list) that users can choose from. The selected value is then sent to the assistant as user input.

#### pause

Pauses before sending the next message to the channel, and optionally sends a "user is typing" event (for channels that support it).

#### speech\_to\_text

Sends a command to the Speech to Text service instance used by the phone integration. These commands can dynamically change the configuration or behavior of the service during a conversation.

#### start\_activities

Sends a command to a channel integration to start one or more activities that are specific to that channel. You can use this response type to restart any activity you previously stopped using the `stop_activities` response type.

#### stop\_activities

Sends a command to a channel integration to stop one or more activities that are specific to that channel. The activities remain stopped until they are restarted using the `start_activities` response type.

#### text

Displays text (or reads it aloud, for the phone integration). To add variety, you can specify multiple alternative text responses. If you specify multiple responses, you can choose to rotate sequentially through the list, choose a response randomly, or output all specified responses.

#### text\_to\_speech

Sends a command to the Text to Speech service instance used by the phone integration. These commands can dynamically change the configuration or behavior of the service during a conversation.

#### user\_defined

A custom response type containing any JSON data the client or integration knows how to handle.

#### video

Displays a video specified by a URL.

## Channel integration support for response types

The following table indicates which channel integrations support each type. For additional information about any channel-specific limitations, see [Response types reference](#).

Response type	Web chat	Phone	SMS	Slack	Facebook	WhatsApp
audio	✓	✓	✓	✓	✓	✓
channel_transfer		✓	✓	✓	✓	✓
connect_to_agent	✓	✓			✓	

date	✓					
dtmf		✓				
end_session		✓				
iframe	✓				✓	
image	✓		✓	✓	✓	✓
option	✓	✓	✓	✓	✓	✓
pause	✓				✓	✓
speech_to_text		✓				
start_activities		✓				
stop_activities		✓				
text	✓	✓	✓	✓	✓	✓
text_to_speech		✓				
user_defined	✓	✓	✓	✓	✓	✓
video	✓		✓	✓	✓	✓

## Dynamic options

An *options* response presents customers with a list of choices to select from. You can use the **dynamic** setting to generate the list from options that might be different each time.

Dynamic options are generated based on the data stored in a variable, which must be available to the step asking the question. The source variable must contain an array of values, each of which represents one of the options that will be presented to the customer. The items in the array can be simple values such as strings or numbers (for example, `[ "Raleigh", "Boston", "New York" ]`) or compound JSON objects.

A common scenario for dynamic options is when an array is returned from an external API that you call using a custom extension. For example, you might use a custom extension to retrieve a list of credit cards associated with a customer's account. You can then use dynamic options to ask the customer which card to use during the conversation. (For more information about custom extensions, see [Calling a custom extension](#).)

Your actions might also populate the source variable using expressions. For example, you might use a session variable to build a shopping cart containing items the customer has decided to purchase. An action for removing an item from the cart could then use dynamic options to show the items in the cart so the customer can select which one to remove. (For more information about using expressions for variable values, see [Using an expression to assign a value to a session variable](#).)

## Defining dynamic options

To define a dynamic options customer response:

1. In a step, click **Define customer response**.
2. Choose the **Options** response type.
3. Click the **Dynamic** toggle.
4. In the **Source variable** field, choose the variable that contains the array that defines the dynamic options (for example, the variable containing the response from a custom extension that you called in a previous step).
5. **Optional:** In the **Option** field, write an expression that maps the items in the source array to the options that will be listed. This expression serves as a template that converts each item in the array to a meaningful value that will be displayed to the customer. In this expression, use the dynamic variable  `${item}` to represent the item.

In some situations, you do not need to specify an expression:

- If the items in the array are simple values such as strings or integers, the value of each item is automatically shown as an option. However, you might still want to define a mapping if you want to manipulate or reformat the items to make them more meaningful. For example, you might use the expression `"Part #"` +  `${item}` to show part numbers using the format `Part #12345`.
- If the items in the array are JSON objects, the default mapping looks for a property called `label` and uses its value (if present) as the option. If the item does not include a `label` property, or you do not want to use the value of the `label` property as the option, you must write an expression to specify a mapping. You can use dot notation to refer to a property in the object using its JSON path (for example,  `${item}.name`).

## Mapping examples

Suppose you want to build an action that shows a list of pets available for adoption and prompts the customer to select a pet to see more information about. The source variable contains an array from a custom extension in the following format:

```
$ [
  {
    "id": "123",
    "name": "Casey",
    "breed": "Shetland sheepdog",
    "age": 3
  },
  {
    "id": "987",
    "name": "Phoebe",
    "breed": "chihuahua",
    "age": 7
  }
]
```

The schema for the items does not include a `label` property, so the default mapping is not available. Instead, you might use an expression to build a complex label that includes data taken from several different properties. For example, you might use the expression  `${item}.name + " (" + ${item}.breed + ", age " + ${item}.age + ")"` to define the option labels:

Which pet do you want to see more information about?

Casey (Shetland sheepdog, age 3)  
Phoebe (chihuahua, age 7)

Remember that you can use expression methods to manipulate values from the source variable in various ways. For example, you might have an action customers use to select a credit card for payment, but for security reasons you don't want to show the entire card number. You could write an expression that uses the `substring()` method to include only the last four digits of

each card number (for example, "Card ending in " + \${item}.card\_number.substring(16, 20)).

## Referencing the selected item

After the customer has selected one of the dynamically generated options, you will probably need to reference the selected item in a subsequent step.

If you reference the action variable representing the customer response, the default is to use the value of the selected option. However, in some situations, you might not want to use the same value that was used to display the option to the customer. Instead, you might need to use a unique identifier or other property that unambiguously identifies the selected option.

For example, if the customer selects a pet to show more information about, you probably need to use a unique identifier (the `id` property in our example) to query the database, since the pet's name, age, and breed might not be unique. Or if the customer is selecting a credit card from options that show only the last four digits, you will need to use the full credit card number to access the account details or complete a transaction.

In this situation, you can write an expression to access the original properties of the selected item:

1. Create or edit a step that comes after the step in which the customer selects from the dynamic options.
2. In the **Variable values** section, write an expression to assign a value to a session variable. (For more information, see [Using an expression to assign a value to a session variable](#).)
3. In the expression editor, type a dollar sign (\$) and then select the step in which the customer selected the dynamic option.
4. Use the property name `item` to represent the selected item, and dot notation to access its properties. For example, the following expression accesses the `id` property of the item selected in a previous step:

```
$ ${step_331}.item.id
```

You can use a complex expression to construct a value using multiple properties of the selected item. For example, you might use an expression such as  `${step_123}.item.firstname + " " + ${step_123}.item.lastname` to construct a person's full name. Use the expression to define the value in whatever format you need to complete any required action.

## Guiding customers with journeys Beta

A journey is an interactive response that you can use to guide your customers through a complex task, or to give them a tour of new features, taking advantage of capabilities your website already supports. A journey is a multipart response that can combine text, video, and images presented in sequence.

This beta feature is available for evaluation and testing purposes but should not be used in production environments. Journeys require web chat version 6.9.0 or later.

When the customer starts a journey, the chat window temporarily closes. The web chat integration then presents the journey elements one step at a time in a small window superimposed over your website, enabling your customers to navigate and use the website as they step through the journey. At any time during the journey, the customer can freely return to the assistant chat window and then resume the journey.

Lendyr Assistant

dispute charge

I can help you understand charges and file a dispute if needed. Do you want me to show you or tell you how to view charge details and file disputes?

Show me Tell me

Show me

**Let's dispute a charge!**

Follow along with this guided journey to learn how to find and dispute charges.

Get started →

?

Type something...

Built with IBM Watson® ⓘ

You might use a journey in situations like the following examples:

- Onboarding new customers to your product or website and showing them where everything is

Lendyr Topaz Card - 7619

**\$1,312.41** Current balance

Pending

Cafe Tropical -\$32.67

April 12, 2022

Rose Apothocary -\$134.98

April 12, 2022

Sirius Cybernetics -\$36.97

Corp April 12, 2022

MomCorp -\$83.07

April 12, 2022

The option to Dispute is marked on the right hand side of each transaction. To file a dispute for a charge, click Dispute.



- Providing customers with step-by-step guidance for a complex task, like filing a claim or creating an account
- Promoting sales opportunities in your product to target users during specific marketing opportunities, such as offering a new rewards program to customers who are concerned about expenses

**Tip:** For more information about deciding when and how to use journeys, see our [best practices guide](#).

## Creating a journey

A journey is defined using the `user_defined` response type, which is available only in the JSON editor. (For more information, see [Defining responses using the JSON editor](#).) To create a journey, follow these steps:

1. In the action editor, create or edit the step from which you want to start the journey.
2. Click the **Switch to JSON editor** icon  to open the JSON editor.
3. In the `generic` array, create a `user_defined` response. (For more information, see [Defining responses using the JSON editor](#).)

A journey is defined using the following structure:

```
$ "user_defined": {
  "user_defined_type": "IBM_BETA_JOURNEYS TOUR",
  "skip_card": true|false,
  "card_title": "{title}",
  "card_description": "{description}",
  "steps": [
    ...
  ]
}
```

where:

`user_defined_type`

The specific type of user-defined response you are defining. To define a journey, always set this property to `IBM_BETA_JOURNEYS TOUR`.

`skip_card`

An optional property that specifies whether the web chat should start the journey immediately without waiting for the customer to click the introductory card in the web chat window. (The default value is `false`.)

**Tip:** You can use this option to start a journey directly from your website, even if the web chat is not open. For more information, see [Starting a journey without opening the web chat](#).

`card_title`

The title to display on the introductory card that appears in the web chat when a journey is available (for example, `Website tour` or `Disputing a charge`).

`card_description`

The description to display on the introductory card. Describe the journey so your customers can decide whether they want to open it.

## steps

An array of responses defining the steps in the journey.

### Defining steps

Each step in a journey is defined as a JSON object describing a response to be shown to the customer, using a format that is similar to how you define assistant responses directly in the `generic` array. Steps in a journey are shown to the customer one at a time, in the order in which you list them in the `steps` array.

As with assistant responses, the `response_type` property identifies the type of response:

#### text

A step that shows only text.

```
$ {  
  "response_type": "text",  
  "text": "This is the text of the response."  
}
```

Markdown formatting and links are supported in `text` steps. For more information, see [Markdown formatting](#).

**⚠️ Important:** Note that the structure of a `text` step in a journey is different from the `text` response type for assistant responses. Instead of an array of text values, only a single `text` component is supported.

#### image

A step that shows an image, along with an optional description.

```
$ {  
  "response_type": "image",  
  "source": "https://example.com/image.png",  
  "description": "This is the description of the image."  
}
```

The `source` property must be the `https:` URL of a publicly accessible image. The specified image must be in `.jpg`, `.gif`, or `.png` format.

#### video

A step that shows a video, along with an optional description.

```
$ {  
  "response_type": "video",  
  "source": "https://example.com/videos/example-video.mp4",  
  "description": "This is the description of the video."  
}
```

The URL specified by the `source` property can be either of the following:

- The URL of a video file in a standard format such as MPEG or AVI. In the web chat, the linked video will render as an embedded video player.

HLS (`.m3u8`) and DASH (MPD) streaming videos are not supported.

- The URL of a video hosted on a supported video hosting service. In the web chat, the linked video will render using the embeddable player for the hosting service.

Specify the URL you would use to view the video in your browser (for example, `https://www.youtube.com/watch?v=52bpMKVigGU`). You do not need to convert the URL to an embeddable form; the web chat will do this automatically.

You can embed videos hosted on the following services:

- [YouTube](#)
- [Facebook](#)
- [Vimeo](#)
- [Twitch](#)
- [Streamable](#)
- [Wistia](#)
- [Vidyard](#)

## Example

The following example defines a journey that shows users how to dispute a charge, using a combination of text, image, and video responses.

```
$ {
  "generic": [
    {
      "response_type": "user_defined",
      "user_defined": {
        "card_title": "Let's dispute a charge!",
        "card_description": "Follow along with this guided journey to learn how to find and dispute charges.",
        "user_defined_type": "IBM_BETA_JOURNEYS TOUR",
        "steps": [
          {
            "response_type": "text",
            "text": "Charges are listed on the Transactions page. **Click your profile photo** in the top right corner of your screen, and then **click Transactions** from the menu."
          },
          {
            "response_type": "text",
            "text": "Here you can view your charges.\n**Scroll through the Transactions page and review your charges.** Each charge contains a merchant name, transaction date, and amount charged."
          },
          {
            "response_type": "image",
            "source": "https://example.com/image.png",
            "alt_text": "Image showing location of Dispute option",
            "description": "The option to Dispute is marked in red on the right hand side of each row in the Transactions table. Just click here to file a dispute."
          },
          {
            "response_type": "video",
            "source": "https://vimeo.com/769580398",
            "description": "Watch this short video to learn what to expect now that you've filed a dispute."
          }
        ]
      }
    ]
  ]
}
```

## Starting a journey without opening the web chat

Although journeys are part of the web chat integration, you can make it possible for your customers to start a journey directly from your website without opening the web chat window at all. For example, you might want to include a **Show me** button on your website that customers can click to launch an interactive tour of the page.

To start a journey without opening the web chat:

1. In the action that sends the journey response, edit the JSON that defines the journey. Include `"skip_card": true` to bypass the introductory card.
2. On your website, use the `send()` instance method to send a message to the assistant that triggers the action that starts the journey (such as `Give me a tour`). Send the message in response to whatever event you want to use to trigger the journey (such as a button click or page load).

Your customers can now start the interactive journey directly from your website without having to open the web chat first. (If the web chat window is opened later, the introductory card for the journey appears in the chat history.)

## Limitations

This beta feature currently has the following limitations:

- The preview pane does not support journeys. If you want to preview a journey, use the shareable preview link. (For more information about the preview link, see [Copying a link to share](#).)
- Journeys currently do not meet accessibility requirements.
- Journeys are not supported if you are using the `element configuration option` to render the web chat in a custom DOM element.
- When the customer starts a journey, the web chat window temporarily closes, but will reopen when the journey finishes. If you are using the `window:close` event to trigger the display of a post-chat form, your code should check the value of the new `event.reason` parameter of the event and verify that it is not set to `open_tour`.

## Writing expressions

You can write *expressions* to specify values that are independent of, or derived from, values that are collected in steps or stored in session variables. You can use an expression to define a step condition or to define the value of a session variable.

### Using an expression in a step condition

You can use an expression in a step condition if you want to condition a step on the result of a calculation based on information you have gathered during the conversation.

For example, suppose a customer has \$200 in a savings account and wants to transfer \$150 from it to a new checking account. The funds transfer fee is \$3, and the bank charges a fee when a savings account contains less than \$50. You could create a step with a step condition that checks for this situation. The step condition would use an expression like this:

```
 ${savings} - (${Step_232} + ${transfer_fee}) < 50
```

where:

- `${savings}` represents a session variable that stores the customer's savings account total.

- `${Step_232}` represents the step that asks for the amount the customer wants to transfer.
- `${transfer_fee}` represents a session variable that specifies the fee for a funds transfer.

If the step condition is met, the step warns the user that the requested transfer will bring the savings account balance below the \$50 minimum and incur a fee, and ask to confirm before proceeding.

To use an expression in a step condition, follow these steps:

1. From the step, click **Add condition**.

A condition is generated automatically with the most likely choice, which is typically any variables that were set in the previous step.

2. Click the first segment of the generated condition, and then scroll down and click **Expression**.

3. **Optional:** Click the  **Expand** icon to open the expression editor window. (You can also type the expression directly in the field without opening the window, but the editor makes it easier to edit a longer or more complex expression.)

4. Type the expression that you want to use.

## Using an expression to assign a value to a session variable

You can use an expression when assigning a value to a session variable if you want the variable's value to be calculated based on other variables.

For example, suppose you want to tell your customer the total cost of a purchase, including 6% sales tax and a flat \$3.00 processing fee. To calculate the total cost, you could create a session variable and assign the value using an expression:

```
$ (${price} * 1.06) + 3
```

You can then reference this variable in the **Assistant says** field.

To use an expression when assigning a value to a session variable, follow these steps:

1. From within a step, choose the **Set variable values**  icon.

2. Click **Set new value**.

3. From the drop-down list, select the session variable you want to store the value in.

4. After **to**, select **Expression**.

5. Type the expression you want to use.

6. If you are using the expression editor, click **Apply** to save your changes and close the editor window.

**Tip:** You can also use an expression to assign an initial value to a session variable. In the **Session variable** window, go to the **Initial value** field and click **Use expression**.

## Expression syntax

The Watson Assistant expression language is based on the Spring Expression Language (SpEL), but with some important differences in syntax. For detailed background information about SpEL, see [Spring Expression Language \(SpEL\)](#).

## Variables

To reference a variable in an expression, type a dollar sign (\$) and then select a variable from the list. The reference is inserted into your expression in the correct notation, referencing the variable using its variable ID rather than its display name (for example, \${step\_773} or \${customer\_id}). Do not edit this reference unless you want to refer to a different variable and you are sure of its variable ID.

## Standard math

For numeric values, you can use expressions to perform mathematical calculations. For basic arithmetic, you can use standard operators (+, -, \*, /).

You can also use methods to perform additional mathematical operations. For more information, see [Expression language methods for actions](#).

## Arrays

To define an array value, type the value using square brackets, with commas separating the items (for example, [ "one", "two", "three" ]).

To reference an item in an array, use bracket notation and specify the zero-based index of the item in the array. For example, \${Items}[0] represents the first item in the array Items.

**Tip:** You can also use the array method `get()` to retrieve an item from an array. For more information, see [Expression language methods for actions](#).

## JSON objects

Use JSON notation to define compound objects in expressions. For example, the following expression assigns a complex JSON object as the value for a variable:

```
$ {
  "name": {
    "firstname": "John",
    "lastname": "Doe"
  },
  "age": 36
}
```

You can use variables and standard math within JSON to create dynamic objects that are calculated at run time. For example, the following expression defines a JSON object that references variables and calculates an average value:

```
$ {
  "temp_1": ${temp_1},
  "temp_2": ${temp_2},
  "avg_temp": (${temp_1} + ${temp_2}) / 2
}
```

To refer to a child object contained in a JSON value, use dot notation to express the path to the object (for example, \${customer}.name.lastname).

If you need to refer to a child of an object that might or might not be defined, use the safe navigation operator (?). For example, the expression \${customer}.name?.lastname evaluates to null if customer.name is null. (Without the safe navigation operator, an error would result.)

## Methods

Use expression language methods to manipulate values (for example, formatting a string or appending an item to an array). For

more information about the supported methods for each data type, see [Expression language methods for actions](#).

## Customizing channels

### Handling phone interactions

If your assistant uses the phone integration, you can use various response types to customize the behavior of the integration or manage the flow of conversations that your assistant has with customers over the telephone.

You can use response types to perform the following phone-specific actions:

- [Apply advanced settings to the Speech to Text service](#)
- [Apply advanced settings to the Text to Speech service](#)
- [Transfer a call to a live agent](#)
- [Play hold music or a voice recording](#)
- [Enable keypad entry](#)
- [Transfer the conversation to the web chat integration](#)
- [End the call](#)
- [Send a text message during a phone conversation](#)

In some cases, you might want to combine response types to perform multiple actions. For example, you might want to implement two-factor authentication by requesting phone keypad entry and sending a text message from the same action step. For more information, see the following:

- [Define a sequence of phone commands](#)

You can also perform the following phone-specific actions:

- [Inject custom values into CDR log events](#)
- [Access phone integration context variables from your action](#)

For reference information about response types, see [Response types reference](#).

### Adding phone-specific responses to your assistant

To initiate a voice-specific interaction from a an action step, add a response within the `generic` array using the appropriate response type. For more information about using the JSON editor to add responses, see [Defining responses using the JSON editor](#).

### Applying advanced settings to the Speech to Text service

Use the `speech_to_text` response type to send configuration commands to the Speech to Text service instance used by the phone integration. By sending a `speech_to_text` response from an action step, you can dynamically change the Speech to Text configuration during a conversation.

By default, any Speech to Text configuration changes you make persist for the remainder of the conversation, or until you update them again. You can change this behavior by specifying the `update_strategy` property of the `parameters` object.

The format of the `speech_to_text` response type is as follows:

```
{  
  "generic": [  
    {  
      "response_type": "speech_to_text",  
      "command_info": {  
        "type": "<command type>",  
        "parameters": {  
          "key": "value"  
        }  
      }  
    }  
  ]  
}
```

```

        "parameter name1": "parameter value",
        "parameter name2": "parameter value"
    }
}
]
}

```

Each command type along with its related parameters are described in the following sections.

`command_info.type : configure`

Dynamically reconfigures the Speech to Text service by applying a set of configuration parameters, which can be based on the conversation flow. For example, you might want to choose a particular customization ID or grammar at a specific point in the conversation.

parameter	description	required	default
<code>narrowband_recognize</code>	The Speech to Text service configuration to use for narrowband codecs (such as PCMU and PCMA, which are sampled at 8 kHz). The parameters defined by this object are used when connecting to the Speech to Text service for speech recognition requests. For more information about these parameters, see the <a href="#">Speech to Text API documentation</a> .	no	Current Speech to Text configuration
<code>broadband_recognize</code>	The Speech to Text service configuration to use for broadband codecs (such as G722, which is sampled at 8 kHz). The parameters defined by this object are used when connecting to the Speech to Text service for speech recognition requests. For more information about these parameters, see the <a href="#">Speech to Text API documentation</a> .	no	Current Speech to Text configuration
<code>band_preference</code>	Specifies which audio band (narrowband or broadband) is preferred when negotiating audio codecs for the session. Set to <code>broadband</code> to use broadband audio when possible.	no	<code>narrowband</code>
<code>update_strategy</code>	Specifies the update strategy to use when setting the speech configuration. Possible values include: <ul style="list-style-type: none"> <li><code>replace</code>: Replaces the configuration for the rest of the session. Any root-level fields in the new configuration completely overwrite the previous configuration.</li> <li><code>replace_once</code>: Replaces the configuration only for the next turn of the conversation. Subsequently, the previous configuration is used.</li> <li><code>merge</code>: Merges the new configuration with the existing configuration for the rest of the session. Only changed parameters are overwritten; any other configuration parameters are unchanged.</li> <li><code>merge_once</code>: Merges the new configuration with the existing configuration only for the next turn of the conversation. Subsequently, the previous configuration is used.</li> </ul>	no	<code>replace</code>

The parameters that you can set for `narrowband_recognize` and `broadband_recognize` reflect the parameters that are made available by the Speech to Text WebSocket interface. The WebSocket API sends two types of parameters: query parameters, which are sent when the phone integration connects to the service, and message parameters, which are sent as part of the JSON data in the request body. For example, `model` is a query parameter, and `smart_formatting` is a WebSocket message parameter. For a full list of parameters, see the [Speech to Text API documentation](#).

You can define the following query parameters for the phone integration's connection to the Speech to Text service. Any other

parameter that you define for `narrowband` or `broadband` is passed through as part of the WebSocket message request.

- `model`
- `acoustic_customization_id`
- `version`
- `x-watson-learning-opt-out`
- `base_model_version`
- `language_customization_id`

The following parameters from the Speech to Text service can't be modified because they have fixed values that are used by the phone integration.

- `action`
- `content-type`
- `interim_results`
- `continuous`
- `inactivity_timeout`

 **Note:** When configuring dynamically from Watson Assistant using the `configure` command, note that only the root level fields, such as `narrowband` or `broadband`, are updated. If these fields are omitted from the command, the original configuration settings persist. You can use the `update_strategy` values `merge` and `merge_once` to merge configuration parameters with the existing configuration.

## Using a custom language model

When you set up the phone integration, you can configure the integration to use a custom language model all the time.

However, you might want to use a standard language model most of the time, and specify a custom language model to use only for specific topics that your assistant is designed to help customers with. For example, you might want to use a custom model that specializes in medical terms for an action that helps with medical bills only. You can apply a custom language model for a specific branch of the conversation.

For more information, about custom language models, see [Creating a custom language model](#).

To apply a custom language model to an action step, use the `speech_to_text` response type.

```
{  
  "generic": [  
    {  
      "response_type": "speech_to_text",  
      "command_info": {  
        "type": "configure",  
        "parameters": {  
          "narrowband_recognize": {  
            "x-watson-learning-opt-out": true,  
            "model": "en-US_NarrowbandModel",  
            "profanity_filter": true,  
            "smart_formatting": true,  
            "language_customization_id": "81d3630-ba58-11e7-aa4b-41bcd3f6f24d",  
            "acoustic_customization_id": "e4766090-ba51-11e7-be33-99bd3ac8fa93"  
          }  
        }  
      }  
    }  
  ]  
}
```

You can also apply an acoustic model that you might have trained to deal with background noise, accents, or other things that

are associated with the quality or noise of the signal.

## Using a custom grammar

The Speech to Text service supports the use of grammars. A grammar allows you to configure the audio to match specific characteristics only.

You can think of it this way:

- A custom language model expands the service's base vocabulary.
- A grammar restricts the words that the service can recognize from that vocabulary.

When you use a grammar with a custom language model for speech recognition, the service can recognize only words, phrases, and strings that are recognized by the grammar. For example, maybe you want to accept only a yes or no response. You can define a grammar that allows only those options.

For more information, see [Using grammars with custom language models](#).

This example shows how to specify a custom grammar during the conversation:

```
{  
  "generic": [  
    {  
      "response_type": "speech_to_text",  
      "command_info": {  
        "type": "configure",  
        "parameters": {  
          "update_strategy": "merge_once",  
          "narrowband_recognize": {  
            "x-watson-learning-opt-out": true,  
            "grammar_name": "names-abnf",  
            "language_customization_id": "81d3630-ba58-11e7-aa4b-41bcd3f6f24d"  
          }  
        }  
      }  
    }  
  ]  
}
```

## Examples

The following examples illustrate how to use the `speech_to_text` response type to send configuration commands to the Speech to Text service.

### Example: Setting the language model

In this example, the language model is switched to Spanish (`es-ES_NarrowbandModel`), and smart formatting is enabled.

```
{  
  "generic": [  
    {  
      "response_type": "speech_to_text",  
      "command_info": {  
        "type": "configure",  
        "parameters": {  
          "narrowband_recognize": {  
            "model": "es-ES_NarrowbandModel",  
            "smart_formatting": true  
          }  
        }  
      }  
    }  
  ]  
}
```

```
    ]  
}
```

## Example: Updating the `recognizeBody` property for one conversation turn

The following example shows how to specify the use of a custom language model for a single turn of the conversation turn. This is done by setting `update_strategy` to `merge_once` and specifying the the ID of the custom language model in the configuration parameters.

```
{  
  "generic": [  
    {  
      "response_type": "speech_to_text",  
      "command_info": {  
        "type": "configure",  
        "parameters": {  
          "update_strategy": "merge_once",  
          "narrowband_recognize": {  
            "language_customization_id": "ao45vohgFuxy0QRgztu-02I10ut7aJcM-AdInT-VWgj3V"  
          }  
        }  
      }  
    }  
  ]  
}
```

## Applying advanced settings to the Text to Speech service

Use the `text_to_speech` response type to send configuration commands to the Text to Speech service instance used by the phone integration. By sending a `text_to_speech` response from an action step, you can dynamically change the Text to Speech configuration during a conversation.

By default, any Text to Speech configuration changes you make persist for the remainder of the conversation, or until you update them again. You can change this behavior by specifying the `update_strategy` property of the `parameters` object.

The format of the `speech_to_text` response type is as follows:

```
$  
{  
  "generic": [  
    {  
      "response_type": "text_to_speech",  
      "command_info": {  
        "type": "<command type>",  
        "parameters": {  
          "parameter name": "parameter value",  
          "parameter name": "parameter value"  
        }  
      }  
    }  
  ]  
}
```

Each command type along with its related parameters are described in the following sections.

### **command\_info.type : configure**

Dynamically reconfigures the Text to Speech service by applying a set of configuration parameters, which can be based on the conversation flow. For example, you might want to choose a particular voice at a specific point in the conversation.

parameter	description	required	default
synthesize	The Text to Speech service configuration to use when synthesizing audio. The parameters defined by this object are used when connecting to the Text to Speech service for speech synthesis requests. For more information about these parameters, see the <a href="#">Text to Speech API documentation</a> .	yes	Current Text to Speech configuration
update_strategy	<p>Specifies the update strategy to use when setting the speech configuration. Possible values include:</p> <ul style="list-style-type: none"> <li>• <b>replace</b>: Replaces the configuration for the rest of the session. Any root-level fields in the new configuration completely overwrite the previous configuration.</li> <li>• <b>replace_once</b>: Replaces the configuration only for the next turn of the conversation. Subsequently, the previous configuration is used.</li> <li>• <b>merge</b>: Merges the new configuration with the existing configuration for the rest of the session. Only changed parameters are overwritten; any other configuration parameters are unchanged.</li> <li>• <b>merge_once</b>: Merges the new configuration with the existing configuration only for the next turn of the conversation. Subsequently, the previous configuration is used.</li> </ul>	no	replace

The parameters that you can set for `synthesize` reflect the parameters that are made available by the Text to Speech WebSocket interface. The WebSocket API sends two types of parameters: query parameters, which are sent when phone integration connects to the service, and message parameters, which are sent as part of the JSON data in the request body. For a full list of parameters, see the [Text to Speech API documentation](#).

`command_info.type : disable_barge_in`

Disables speech barge-in so that playback isn't interrupted when the caller speaks while audio is being played back.

No parameters.

`command_info.type : enable_barge_in`

Enables speech barge-in so that callers can interrupt playback by speaking.

No parameters.

## Changing the assistant's voice

You can change the voice of your assistant when it covers certain topics in the conversation that warrant it. For example, you might want to use a voice with a British accent for a branch of the conversation that applies only to customers in the UK.

This example shows how to specify a voice during the conversation:

```
{
  "generic": [
    {
      "response_type": "text_to_speech",
      "command_info": {
        "type": "configure",
        "parameters": {
          "synthesize": {
            "voice": "en-GB_KateV3Voice"
          }
        }
      }
    }
  ]
}
```

```
        }
    ]
}
```

In the `voice` parameter, specify the voice model that you want to use. For more information about voice model options, see [Supported languages and voices](#).

**Note:** The model you specify must be one that is supported by the Text to Speech service instance that is configured for use with the integration.

## Transferring a call to a live agent

When you configure the phone integration, you can optionally set up backup call center support, which makes it possible for the assistant to transfer a call to a human. You can use the *Connect to agent* response type in an action step to initiate a transfer to a live agent at a specific point in the conversation. When a *Connect to agent* response is sent to the phone integration, a SIP transfer is initiated using the SIP `REFER` message, as defined by [RFC 5589](#).

For more information about initiating a transfer to a live agent during the conversation, see the following documentation:

- [Setting up live agent escalation](#)
- [Defining responses using the JSON editor](#)

The phone integration supports additional parameters for the *Connect to agent* response type. You can add these phone-specific parameters to the `connect_to_agent` response type using the JSON editor.

The `connect_to_agent` response type supports the ability to specify the target transfer information under the `transfer_info` parameter.

The following example shows a transfer that uses all of the configurable parameters:

```
{
  "generic": [
    {
      "response_type": "connect_to_agent",
      "transfer_info": {
        "target": {
          "service_desk": {
            "sip": {
              "uri": "sip:user\\@domain.com",
              "transfer_headers": [
                {
                  "name": "Customer-Header1",
                  "value": "Some-Custom-Info"
                },
                {
                  "name": "User-to-User",
                  "value": "XXXXXX"
                }
              ],
              "transfer_headers_send_method": "refer_to_header"
            }
          }
        }
      },
      "agent_available": {
        "message": "I'll transfer you to an agent"
      },
      "agent_unavailable": {
        "message": "Sorry, I could not find an agent."
      }
    }
  ]
}
```

```

    },
    "message_to_human_agent": "The caller needs help resetting their password"
}
]
}

```

The `connect_to_agent` response type supports the following phone-specific properties.

Parameter	Default	Description
<code>service_desk.sip.uri</code>	N/A	The SIP or telephone URI to transfer the call to, such as <code>sip:12345556789\@myhost.com</code> or <code>tel:+18883334444</code> . Optional when using the <code>hangup</code> method.
<code>service_desk.sip.transfer_method</code>	<code>refer</code>	Determines how to transfer the call: <ul style="list-style-type: none"> <li><code>refer</code>: The call is transferred by sending a SIP REFER request. This is the default value.</li> <li><code>hangup</code>: The call is transferred by sending a SIP BYE request.</li> </ul>
<code>service_desk.sip.transfer_target_header</code>	<code>Transfer-Target</code>	The SIP header that contains the transfer target when a BYE request is used for transferring the call. This option is supported only in the <code>hangup</code> method.
<code>service_desk.sip.transfer_headers</code>	N/A	A list of custom header field name/value pairs to be added to a transfer request
<code>service_desk.sip.transfer_headers_send_method</code>	<code>custom_header</code>	The method by which the SIP transfer headers are sent: <ul style="list-style-type: none"> <li><code>custom_header</code>: Sends the transfer headers as part of the SIP message. This is the default value.</li> <li><code>contact_header</code>: Sends the transfer headers in the <code>Contact</code> header. This option is not supported in the <code>hangup</code> method.</li> <li><code>refer_to_header</code>: Sends the transfer headers in the <code>Refer-To</code> header. This option is not supported in the <code>hangup</code> method.</li> </ul>

If you define a SIP URI as the transfer target, escape the at sign (@) in the URI by adding two backslashes (\\\) in front of it. This is to prevent the string from being recognized as part of the entity shorthand syntax.

```
"uri": "sip:12345556789\\@myhost.com"
```

## Transferring after hangup

By default, the phone integration transfers calls by using a SIP `REFER` request. Depending on the IVR service provider, you might need to configure call transfer to use a SIP `BYE` request instead. Use the `transfer_method` attribute to specify how to transfer the call, using either `refer` or `hangup`. When `transfer_method` is set to `hangup` instead of `refer`, the behavior of the transfer action changes. Instead of sending a SIP `REFER` request, the phone integration plays back any associated text and

then hangs up the call by sending a SIP `BYE` request.

After the hangup, the phone integration passes the transfer destination that is specified in the `url` attribute to the call anchor in the `BYE` message. The header field that contains the transfer target is determined by the `transfer_target_header` attribute. If the `transfer_target_header` attribute isn't specified, the phone integration uses `Transfer-Target`.

```
{  
  "generic": [  
    {  
      "response_type": "connect_to_agent",  
      "transfer_info": {  
        "target": {  
          "service_desk": {  
            "sip": {  
              "uri": "sip:user\\@domain.com",  
              "transfer_method": "hangup",  
              "transfer_target_header": "Transfer-Target"  
            }  
          }  
        }  
      },  
      "agent_available": {  
        "message": "Please hold on while I connect you with a live agent."  
      },  
      "agent_unavailable": {  
        "message": "Sorry, I could not find an agent."  
      },  
      "message_to_human_agent": "The caller needs help resetting their password"  
    }  
  ]  
}
```

## Transferring upon failure

To configure transfer on failures, go to the **Advanced** tab in the phone integration settings. The following selections can be configured:

- **Transfer failure message**
- **Disconnect call on transfer failure**

For more information, see [Handling call and transfer failures](#).

## Passing Watson Assistant Metadata in SIP Signaling

To support loading the conversational history between the caller and Watson Assistant, the phone integration specifies a value for the `User-to-User` header as a key that can be used with the web chat integration. If `User-to-User` is specified in the `transfer_headers` list, the session history key is sent in the `X-Watson-Assistant-Session-History-Key` header.

The value of the SIP header is limited to 1024 bytes.

How this data is presented in the SIP `REFER` message also depends on the value of `transfer_headers_send_method` (as defined in [Generic Service Desk SIP Parameters](#)).

The following example shows the data included as headers:

```
REFER sip:b@atlanta.example.com SIP/2.0  
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293940223  
To: <sip:b@atlanta.example.com>  
From: <sip:a@atlanta.example.com>;tag=193402342  
Call-ID: 898234234@agenta.atlanta.example.com  
CSeq: 23 REFER
```

```
Max-Forwards: 7
Refer-To: sip:user@domain.com
X-Watson-Assistant-Token: 8f817472-8c57-4117-850d-fdf4fd23ba7
User-to-User: 637573746f6d2d757365722d746f2d75736572;encoding=hex
Contact: sip:a@atlanta.example.com
Content-Length: 0
```

If a custom `User-to-User` header is specified, then the session history key is set in the `X-Watson-Assistant-Session-History-Key` header:

```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293940223
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 93809823 REFER
Max-Forwards: 70
Refer-To: sip:user@domain.com
User-to-User: 637573746f6d2d757365722d746f2d75736572;encoding=hex
X-Watson-Assistant-Session-History-Key: dev::latest::212033::0a64c30d-c558-4055-85ad-
ef75ad6cc29d::978f1fd7-4e24-47d8-adb0-24a8a6eff69e::b5ffd6c2-902f-4658-b586-
e3fc170a6cf3::7ad616a350cc48078f17e3ee3df551de
Contact: sip:a@atlanta.example.com
Content-Length: 0
```

This example shows the metadata passed into the `Refer-To` header as query parameters (as defined by [SIP RFC 3261](#)).

```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293940223
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 23 REFER
Max-Forwards: 70
Refer-To: sip:user@domain.com?User-to-User=637573746f6d2d757365722d746f2d75736572%3Bencoding%3Dhex
Contact: sip:a@atlanta.example.com
Content-Length: 0
```

If a custom `User-to-User` header is specified, then the session history key is set in the `X-Watson-Assistant-Session-History-Key` header.

```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293940223
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 93809823 REFER
Max-Forwards: 70
Refer-To: sip:user@domain.com?User-to-User=637573746f6d2d757365722d746f2d75736572%3Bencoding%3Dhex&X-
Watson-Assistant-Session-History-Key=dev::latest::893499::dff9c274-adc4-4f63-93de-
781166760bf8::978f1fd7-4e24-47d8-adb0-24a8a6eff69e::b5ffd6c2-902f-4658-b586-
e3fc170a6cf3::7ad616a350cc48078f17e3ee3df551de
Contact: sip:a@atlanta.example.com
Content-Length: 0
```



**Note:** For Twilio Flex, the `User-to-User` header will use `encoding=ascii`.

## Playing hold music or a voice recording

To play hold music or to play a recorded message, use the `audio` response type. For more information about using response types, see [Defining responses using the JSON editor](#).

You cannot play hold music during a call transfer. However, you might want to play hold music if your assistant needs time to perform processing of some kind, such as calling a client-side action or making a call to a webhook.

The phone integration supports the following properties for the `audio` response type:

Property	Description
<code>source</code>	The URL of a publicly-accessible .wav audio file. The audio file must be single channel (mono) and PCM-encoded, and must have an 8,000 Hz sampling rate with 16 bits per sample.
<code>channel_options.voice_telephony.loop</code>	Whether to repeatedly restart the audio playback after it finishes. The default value is <code>false</code> .

If you set `channel_options.voice_telephony.loop` to `true`, add a user-defined response with the `vgwActForceNoInputTurn` command. This command instructs the phone integration to initiate a turn with a `vgwNoInputTurn` text without waiting for an input from the caller. In the `vgwNoInputTurn` turn you can initiate a transaction while the caller is on hold. When the `vgwNoInputTurn` turn completes, the looped audio stops.

The following example shows an `audio` response with `loop=true`, and a `user_defined` response specifying the `vgwActForceNoInputTurn` command.

```
{
  "generic": [
    {
      "response_type": "user_defined",
      "user_defined": {
        "vgwAction": {
          "command": "vgwActForceNoInputTurn"
        }
      }
    },
    {
      "response_type": "audio",
      "source": "https://upload.wikimedia.org/wikipedia/commons/d/d8/Random_composition3.wav",
      "channel_options": {
        "voice_telephony": {
          "loop": true
        }
      }
    }
  ]
}
```

## Enabling keypad entry

If you want customers to be able to send information by typing it on their phone keypad instead of speaking, you can add support for phone keypad entry. The best way to implement this type of support is to enable dual-tone multifrequency (DTMF) signaling. DTMF is a protocol used to transmit tones that are generated when a user presses keys on a push-button phone. The tones have a specific frequency and duration that can be interpreted by the phone network.

To start listening for tones as the user presses phone keys, use the `dtmf` response type in an action step. This response type can be added using the JSON editor.

```
{
  "generic": [
    {
      "response_type": "dtmf",
      "command_info": {
        "type": "<command type>",
        "value": "<value>"
      }
    }
  ]
}
```

```

  "parameters": {
    "parameter name": "parameter value",
    "parameter name": "parameter value"
  },
  "channels": [
    {
      "channel": "voice_telephony"
    }
  ]
}

```

The `command_info` property specifies a DTMF command for the phone integration. The supported commands and their related parameters are as follows.

`command_info.type : collect`

Instructs the phone integration to collect dual-tone multi-frequency signaling (DTMF) input from a user. This command supports the following parameters:

parameter name	description	required	default
<code>termination_key</code>	The DTMF termination key, which signals the end of DTMF input (for example, #).	no	n/a
<code>count</code>	The number of DTMF digits to collect. This must be a positive integer no larger than 100.	Required if <code>termination_key</code> , or <code>minimum_count</code> and <code>maximum_count</code> , are not defined	n/a
<code>minimum_count</code>	The minimum number of DTMF digits to collect. This property is used along with <code>maximum_count</code> to define a range for the number of digits to collect. This value must be a positive integer with a minimum value of 1 and a maximum value less than <code>maximum_count</code> .	Required if <code>termination_key</code> and <code>count</code> are not defined.	n/a
<code>maximum_count</code>	The maximum number of DTMF digits to collect. This property is used along with <code>minimum_count</code> to define a range for the number of digits to collect. When this number of digits is collected, a conversation turn is initiated. This value must be a positive integer no greater than 100.	Required if <code>termination_key</code> and <code>count</code> are not defined.	n/a
<code>inter_digit_timeout_count</code>	The amount of time (in milliseconds) to wait for a new DTMF digit after a DTMF digit is received. During an active DTMF collection, this timeout activates when the first DTMF collection is received. When the inter-digit timeout is active, it deactivates the post-response timeout timer. If the <code>inter_digit_timeout_count</code> parameter is not specified, the post-response timer resets after receiving each DTMF digit, and it stays active until either the post-response timeout count is met or the collection completes. This value is a positive integer no higher than 100,000 (or 100 seconds).	no	n/a
<code>ignore_speech</code>	Whether to disable speech recognition during collection of DTMF digits, until either the collection completes or a timeout occurs. If this parameter is <code>true</code> , speech recognition is disabled automatically when the first DTMF signal is received.	no	false

<code>stop_after_collection</code>	Whether to stop DTMF input when the DTMF collection completes. After this command, all DTMF input is ignored until it is reenabled using the <code>start</code> response type.	no	false
------------------------------------	--	----	-------

#### `command_info.type : disable_barge_in`

Disables DTMF barge-in so that playback from the phone integration is not interrupted when callers press keys. If `disable_barge_in` is enabled, keys pressed during playback are ignored.

This command has no parameters.

#### `command_info.type : enable_barge_in`

Enables DTMF barge-in so that callers can interrupt playback from the phone integration by pressing a key.

This command has no parameters.

#### `command_info.type : send`

Sends DTMF signals using the phone integration.

This command supports the following parameters:

parameter	description	required	default
<code>digits</code>	An array of JSON objects where each element represents a DTMF tone to be sent to a caller.	yes	n/a
<code>digits[] .code</code>	The event code to send. In addition to the digits 0 through 9, you can specify the following codes: <ul style="list-style-type: none"> <li>• 10: *</li> <li>• 11: #</li> <li>• 12: A</li> <li>• 13: B</li> <li>• 14: C</li> <li>• 15: D</li> </ul>	yes	n/a
<code>digits[] .duration</code>	The duration (in milliseconds) of the event.	no	200
<code>digits[] .volume</code>	The power level of the tone, in dBm0. The supported range is 0 to -63 dBm0.	no	0
<code>send_interval</code>	An interval (in milliseconds) to wait before sending the next DTMF tone in the list.	no	200

## Examples

This example shows the `dtmf` response type with the `collect` command, used to collect DTMF input.

```
{
  "generic": [
    {
      "response_type": "dtmf",
      "command_info": {

```

```

    "type": "collect",
    "parameters": {
        "termination_key": "#",
        "count": 16,
        "ignore_speech": true
    },
    "channels": [
        {
            "channel": "voice_telephony"
        }
    ]
}

```

This example shows the `dtmf` response type with the `send` command, used to send DTMF signals.

```

{
  "generic": [
    {
      "response_type": "dtmf",
      "command_info": {
        "type": "send",
        "parameters": {
          "digits": [
            {
              "code": "9",
              "volume": -8
            },
            {
              "code": "11"
            }
          ],
          "send_interval": 100
        }
      },
      "channels": [
        {
          "channel": "voice_telephony"
        }
      ]
    }
  ]
}

```

## Transferring the conversation to the web chat integration

You can transfer the caller from the current phone call to a [web chat](#) session by using the `channel_transfer` response type.

The assistant sends an SMS message to the caller that includes a URL that the caller can tap to load the web chat widget in the phone's browser. The web chat session displays the history of the phone call and can start the process of collecting information needed to complete the transaction.

This can be useful in situations where the customer can provide information more easily in writing than by speaking (for example, changing an address).

After the transfer has successfully completed, the caller can hang up the phone and continue the conversation using the web chat. The `channel_transfer` response type can be used with the phone integration only if the [SMS with Twilio](#) integration is also configured for the assistant.

```

{
  "generic": [
    {

```

```

  "response_type": "text",
  "values": [
    {
      "text": "I will send you a text message now with a link to our website."
    }
  ],
  "selection_policy": "sequential"
},
{
  "response_type": "channel_transfer",
  "message_to_user": "Click the link to connect with an agent using our website.",
  "transfer_info": {
    "target": {
      "chat": {
        "url": "https://example.com/webchat"
      }
    }
  }
}
]
}

```

## Ending the call

You can instruct your assistant end a phone call by using the `end_session` response type, as shown in this example.

```
{
  "generic": [
    {
      "response_type": "end_session"
    }
  ]
}
```

You can optionally include custom headers to include with the SIP `BYE` request that is generated when the phone integration receives this response type.

This example shows the `end_session` response type with custom SIP headers:

```
{
  "generic": [
    {
      "response_type": "end_session",
      "channel_options": {
        "voice_telephony": {
          "sip": {
            "headers": [
              {
                "name": "Customer-Header1",
                "value": "Some-Custom-Info"
              },
              {
                "name": "User-to-User",
                "value": "XXXXXX"
              }
            ]
          }
        }
      }
    }
  ]
}
```

## Sending a text message during a phone conversation

There are some situations when it is useful to be able to send a text message during an ongoing voice. For example, you might want the customer to specify a street address, which is easier to communicate accurately in writing than by transcribing voice input.

 **Note:** Before you can send SMS messages during a phone call, you must set up the SMS integration. For more information, see [Integrating with SMS](#).

When you exchange a text with a customer during a conversation, the assistant initiates the SMS message exchange. It sends a text message to the user and asks for the user to respond to it.

To send a specific message from an action step, use the `user_defined` response type with the `vgwActSendSMS` command:

```
{  
  "generic": [  
    {  
      "response_type": "text",  
      "values": [  
        {  
          "text": "I will send you a text message now."  
        }  
      ],  
      "selection_policy": "sequential"  
    },  
    {  
      "response_type": "user_defined",  
      "user_defined": {  
        "vgwAction": {  
          "command": "vgwActSendSMS",  
          "parameters": {  
            "message": "Hey, this is Watson Assistant. To send me your street address, respond to this text message with your address."  
          }  
        }  
      }  
    }  
  ]  
}
```

You can specify any of the following parameters in the `parameters` object:

Parameter	Type	Description
message	string	The text of the SMS message to send. Required.
mediaURL	list	A list of URLs for media files to be sent with the message as MMS attachments. Optional.
tenantPhoneNumber	string	The phone number that is associated with the tenant. The format of the number must match the format that is required by the SMS provider. If no <code>tenantPhoneNumber</code> value is provided, the tenant ID from the phone integration configuration for the active call is used. Optional.
userPhoneNumber	string	The phone number to send the SMS message to. The format of the number must match the format that is required by the SMS provider. If no <code>userPhoneNumber</code> value is provided, the voice caller's phone number from <code>From</code> header of the incoming SIP <code>INVITE</code> request is used. Optional.

If your SMS integration supports more than one SMS phone number, or you are using a SIP trunk different from your SMS provider, be sure to specify the phone number that you want to use to send the text message. Otherwise, the text is sent using

the same phone number that was called.

After the assistant receives an SMS message, a new conversation turn is initiated with the text input `vgwSMSMessage`. This input indicates that a message was received from the caller. The text of the customer's message is included as the value of the `vgwSMSMessage` context variable.

If the assistant is unable to send an SMS message to the caller, a new turn is initiated with the text input `vgwSMSFailed`. This input indicates that an SMS message could not be sent to the caller. You can design your assistant to handle such a failure by creating actions that are triggered by the input text `vgwSMSFailed`.

```
{  
  "input": {  
    "message_type": "text",  
    "text": "vgwSMSMessage"  
  },  
  "context": {  
    "skills": {  
      "main_skill": {  
        "user_defined": {  
          "vgwSMSMessage": "1545 Lexington Ave."  
        }  
      }  
    }  
  }  
}
```

## Defining a sequence of phone commands

If you want to run more than one command in succession, include multiple responses in the `generic` array. These commands are processed in the order in which they are specified in the array.

This example shows two responses: first a text response, followed by an `end_session` response to end the call.

```
{  
  "generic": [  
    {  
      "response_type": "text",  
      "values": [  
        {  
          "text": "Goodbye."  
        }  
      ],  
      "selection_policy": "sequential"  
    },  
    {  
      "response_type": "end_session"  
    }  
  ]  
}
```

## Injecting custom values into CDR log events

If you are using a log webhook to log call detail record (CDR) events, you can use the `cdr_custom_data` context variable to add custom data to logged events. You can use this method to record data during a call (for example, to indicate the completion of a specific tasks).

To log custom CDR data, use the JSON editor to edit the context. Define `cdr_custom_data` as a child of the `context.integrations.voice_telephony` object, as in this example:

```
"context": {  
  "integrations": {  
    "voice_telephony": {  
      "cdr_custom_data": {  
        "key": "value"  
      }  
    }  
  }  
}
```

```
  "cdr_custom_data": {  
    "key1": "value1",  
    "key2": "value2"  
  }  
}  
}
```

The `cdr_custom_data` object can contain any valid JSON data.

When you generate a CDR report, the custom data is included in the `injected_custom_data` field, as in this example:

```
{  
  "payload": {  
    ...  
    "injected_custom_data": {  
      "key1": "value1",  
      "key2": "value2"  
    }  
    ...  
  }  
}
```

For more information about the structure of the CDR log event payload, see [CDR log event reference](#).

## Merging and deleting custom CDR data

Each time the `cdr_custom_data` object is defined by an action, the new data is merged with any previously existing data. New values specified for previously defined properties overwrite the previous values, and any new properties are added; otherwise, the previously defined data is unchanged.

To remove a previously defined property, you must explicitly set it to an empty value, as in this example:

```
"context": {  
  "integrations": {  
    "voice_telephony": {  
      "cdr_custom_data": {  
        "key1": ""  
      }  
    }  
  }  
}
```

## Access phone integration context variables from your action

If you want to access the phone integration context variables, use the JSON editor to edit the context.

The following example shows how to access the user phone number (i.e. the phone number that the call was received from):

```
"context": {  
  "variables": [  
    {  
      "value": {  
        "expression":  
        "${${system_integrations.voice_telephony.private.user_phone_number}.replace('+', '')}"  
      },  
      "skill_variable": "user_phone_number"  
    }  
  ]  
}
```

For more information about the phone integration context variables, see [Phone integration context variables](#).

## Extending your assistant with webhooks

### Administering your instance

---

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

## Making a call before processing a message

A premessage webhook makes a call to an external service or application every time a customer submits input. The external service can process the message before it is processed by your assistant.

Add a premessage webhook to your assistant if you want the webhook to be triggered before each incoming message is processed by your assistant.

 **Important:** If you are using a custom channel, note that the premessage webhook works with the v2 /message API only (stateless and stateful). For more information, see the [API reference](#). All built-in channel integrations use this API.

You can use a premessage webhook to do the following types of things:

- Translate the customer's input to the language that is used by your assistant.
- Check for and remove any personally identifiable information, such as an email address or social security number that a customer might submit.

You can use this webhook in coordination with the postmessage webhook. For example, the postmessage webhook can do things like translate the response back into the customer's native language or add back information that was removed for privacy reasons. For more information, see [Making a call after processing a message](#).



**Note:** For environments where private endpoints are in use, keep in mind that a webhook sends traffic over the internet.

## Defining the webhook

You can define one webhook URL to use for preprocessing every incoming message.

The programmatic call to the external service must meet these requirements:

- The call must be a POST HTTP request.
- The request body must be a JSON object (Content-Type: application/json).
- The call must return in 30 seconds or less.



**Tip:** If your external service supports only GET requests, or if you need to specify URL parameters dynamically at run time, consider creating an intermediate service that accepts a POST request with a JSON payload containing any runtime values. The intermediate service can then make a request to the target service, passing these values as URL parameters, and then return the response to the dialog.



**Important:** Do not set up and test your webhook in a production environment where the assistant is deployed and is interacting with customers.

To add the webhook details, complete the following steps:

1. In your assistant, open the environment where you want to configure the webhook.
2. Click the icon to open the environment settings.
3. On the **Environment settings** page, click **Webhooks > Pre-message webhook**.
4. Set the *Pre-message webhook* switch to **Enabled**.
5. Decide whether to return an error if the webhook call fails.

**Important:** When enabled, everything stops until the preprocessing step is completed successfully.

- If you have a critical preprocessing step that must be taken before you want to allow the message to be processed by the assistant, enable this setting.

Take steps to test the process that you are calling on a regular basis so you will know if it's down, and can change this setting to prevent all of your message calls from failing.

- When this setting is disabled, the assistant ignores any errors it encounters and continues to process the incoming message without taking the preprocessing step. If the preprocessing step is helpful but not critical, consider keeping this setting disabled.

6. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

For example, you might write a Cloud Functions web action that checks whether a message is in a language other than English, and if so, send it to the Language Translator service to convert it to English. Specify the URL for your web action, as

in this example:

```
https://us-south.functions.cloud.ibm.com/api/v1/web/my_org_dev/default/translateToEnglish.json
```

You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.

**⚠ Important:** You cannot use a webhook to call a Cloud Functions action that uses token-based Identity and Access Management (IAM) authentication. However, you can make a call to a Cloud Functions web action or a secured web action.

7. In the **Secret** field, add a private key to pass with the request that can be used to authenticate with the external service.

The key must be specified as a text string, such as `purple unicorn`. The maximum length is 1,024 characters. You cannot specify a context variable.

It is the responsibility of the external service to check for and verify the secret. If the external service does not require a token, specify any string you want. You cannot leave this field empty.

For more information about how this field is used, see [Webhook security](#).

8. In the **Timeout** field, specify the length of time (in seconds) you want the assistant to wait for a response from the webhook before returning an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.

9. In the Headers section, add any headers that you want to pass to the service one at a time by clicking **Add header**.

For example, if the external application that you call returns a response, it might be able to send a response in multiple different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header that indicates that you want the resulting value to be returned in JSON format.

Header name	Header value
Content-Type	application/json

#### Header example

The service automatically sends an `Authorization` header with a JWT; you do not need to add one. If you want to handle authorization yourself, add your own authorization header and it will be used instead.

Your webhook details are saved automatically.

## Testing the webhook

**⚠ Important:** Do extensive testing of your webhook before you enable it for an assistant that is being used in a production environment.

The webhook is triggered when a message is sent to your assistant to be processed.

If you enable the setting that returns an error when the webhook call fails, the processing of the assistant is halted entirely if the webhook encounters any issues. Take steps to test the process that you are calling on a regular basis so you will be alerted if the external service is down, and can take actions to prevent all of the incoming messages from failing to be received.

**Tip:** If you call an Cloud Functions web action, you can use the logging capability in Cloud Functions to help you troubleshoot your code. You can [download the command line interface](#), and then enable logging with the [activation](#)

[polling command](#).

## Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you will know that your webhook has an issue if every test message you submit returns a message such as, `There is an error with the message you just sent, but feel free to ask me something else.` If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Error validating webhook response	The webhook's HTTP response body was not a valid <code>/message</code> body.
422 Webhook responded with [500] status code	There's a problem with the external service you called. The code failed or the external server refused the request.
500 Processor Exception :[connections to all backends failing]	An error occurred in the webhook microservice. It could not connect to backend services.

### Error code details

## Webhook security

To authenticate the webhook request, verify the JSON Web Token (JWT) that is sent with the request. The webhook microservice automatically generates a JWT and sends it in the `Authorization` header with each webhook call. It is your responsibility to add code to the external service that verifies the JWT.

For example, if you specify `purple unicorn` in the `Secret` field, you might add code similar to this:

```
const jwt = require('jsonwebtoken');
...
const token = request.headers.authentication; // grab the "Authentication" header
try {
  const decoded = jwt.verify(token, 'purple unicorn');
} catch(err) {
  // error thrown if token is invalid
}
```

## Request body

It is useful to know the format of the request body of the `/message` webhook so that your external code can process it.

The payload contains the request body of the `/message` (stateful or stateless) v2 API request. The event name `message_received` indicates that the request is generated by the `/message` webhook. For more information about the message request body, see the [API reference](#).

```
{
  "payload" : { Copy of request body sent to /message }
  "event": {
    "name": "message_received"
  }
}
```

## Response body

The service that receives the POST request from the webhook must return a JSON object `Accept: application/json`).

The response body must have the following structure:

```
{  
  "payload": {  
    ...  
  }  
}
```

The `payload` object in the response should contain the `payload` object that was received in the request body. Your code can modify property values in the message payload it received (for example, to update property values, or to add or remove context variables); but the message payload returned to the service must conform to the schema for a request to the `message` method. For more information, see the [API reference](#).

## Example 1

This example shows you how to check the language of the input text, and append the language info to the input text string.

In the premessage webhook configuration page, the following values are specified:

- **URL:** `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/check_language`
- **Secret:** none
- **Header name:** Content-Type
- **Header value:** application/json

The premessage webhook calls an IBM Cloud Functions web action name `check_language`.

The node.js code in the `check_language` web action looks as follows.

```
let rp = require("request-promise");

function main(params) {
  console.log(JSON.stringify(params))
  if (params.payload.input.text !== '') {
    // Send a request to the Watson Language Translator service to check the language of the input text.
    const options = { method: 'POST',
      url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
      auth: {
        'username': 'apikey',
        'password': 'nnn'
      },
      headers: {
        "Content-Type": "text/plain"
      },
      body: [
        params.payload.input.text
      ],
      json: true,
    };
    return rp(options)
      .then(res => {
        params.payload.context.skills["actions skill"].user_defined["language"] =
        res.languages[0].language;
        console.log(JSON.stringify(params))
        //Append "in" plus "the language code" to the input text, surrounded by parentheses.
        const response = {
          body : {
            "text": params.payload.input.text + " (" + res.languages[0].language + ")"
          }
        };
        return rp(response)
          .then(res2 => {
            params.payload.output.text = res2.body.text;
            return params;
          })
        }
      })
    }
  }
}

module.exports = main;
```

```

        payload : {
            input : {
                text : params.payload.input.text + ' ' + '(in ' + res.languages[0].language +
            ')
        },
    },
};

return response;
}
}

return {
    body : params
};
};

```

To test the webhook, click **Preview**. In the preview panel, submit the text `Buenos días`. The assistant probably won't understand the input, and will return the response from your *Anything else* node. However, if you go to the Analytics page of your assistant and open the User conversations page, you can see what was submitted. Check the most recent user conversation. The log will show that the user input is `Buenos días (in es)`. The `es` in parentheses represents the language code for Spanish, so the webhook worked and recognized that the submitted text was a Spanish phrase.

## Example 2

This example shows you how to check the language of the incoming message, and if it's not English, translate it into English before submitting it to the assistant.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks the language of the incoming text. The second action in the sequence translates the text from its original language into English.

In the premessage webhook configuration page, the following values are specified:

- **URL:** `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/translation_sequence`
- **Secret:** none
- **Header name:** Content-Type
- **Header value:** application/json

The node.js code for the first web action in your sequence looks as follows:

```

let rp = require("request-promise");

function main(params) {
    console.log(JSON.stringify(params))
    if (params.payload.input.text !== '') {
        const options = { method: 'POST',
            url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
            auth: {
                'username': 'apikey',
                'password': 'nnn'
            },
            headers: {
                "Content-Type": "text/plain"
            },
            body: [
                params.payload.input.text
            ],
            json: true,
        };
        return rp(options)
    }
}

```

```

    .then(res => {
      //Set the language property of the incoming message to the language that was identified by
      Watson Language Translator.
      params.payload.context.skills["actions skill"].user_defined["language"] =
      res.languages[0].language;
      console.log(JSON.stringify(params))
      return params;
    })
  }
else {
  params.payload.context.skills["actions skill"].user_defined["language"] = 'none'
  return params
}
};


```

The second web action in the sequence sends the text to the Watson Language Translator service to translate the input text from the language that was identified in the previous web action into English. The translated string is then sent to your assistant instead of the original text.

The node.js code for the second action in your sequence looks as follows:

```

let rp = require("request-promise");

function main(params) {
  console.log(JSON.stringify(params))
  //If the incoming message is not null and is not English, translate it.
  if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') &&
  (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
    const options = { method: 'POST',
      url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-
      b693-3bc63869nnnn/v3/translate?version=2018-05-01',
      auth: {
        'username': 'apikey',
        'password': 'nnn'
      },
      headers: {
        "Content-Type": "application/json"
      },
      //The body includes the parameters that are required by the Language Translator service, the
      text to translate and the target language to translate it into.
      body: {
        text: [
          params.payload.input.text
        ],
        target: 'en'
      },
      json: true
    };
    return rp(options)
    .then(res => {
      params.payload.context.skills["actions skill"].user_defined["original_input"] =
      params.payload.input.text;
      const response = {
        body : {
          payload : {
            "context" : params.payload.context,
            "input" : {
              "text" : res.translations[0].translation,
              "options" : {
                "export" : true
              }
            },
            ...
          },
          ...
        };
      return response
    })
  }
};


```

```
    }
    return {
      body : params
    }
};
```

When you test the webhook in the preview panel, you can submit `Buenos días`, and the assistant responds as if you said `Good morning` in English. In fact, when you check the *Analytics>User conversations* page, the log shows that the user input was `Good morning`.

You can add a postmessage webhook to translate the message's response back into the customer's native language before it is displayed. For more information see [Example 2](#).

## Removing the webhook

If you decide you do not want to preprocess customer input with a webhook, complete the following steps:

1. From the assistant overview page, click the `:` icon, and then choose **Settings**.
2. Click **Webhooks > Pre-message webhook**.
3. Do one of the following things:
  - To change the webhook that you want to call, click **Delete webhook** to delete the currently specified URL and secret. You can then add a new URL and other details.
  - To stop calling a webhook to process every incoming message, click the *Pre-message webhook* switch to disable the webhook altogether.

## Making a call after processing a message

A postmessage webhook makes a call to an external service or application every time a response is rendered by the assistant. The external service can process the assistant output before it is sent to the channel.

You can add a postmessage webhook to your assistant if you want the webhook to be triggered before each message response is shown to the customer.

 **Important:** If you are using a custom channel, note that the postmessage webhook works with the `v2 /message` API only (stateless and stateful). For more information, see the [API reference](#). All built-in channel integrations use this API.

You can use a postmessage webhook to do things like extract custom responses from an external content repository. For example, you can define actions with custom IDs in the responses instead of text. The postmessage webhook can pass these IDs to an external database to retrieve stored text responses.

You can use this webhook in coordination with the premessage webhook. For example, if you use the premessage webhook to strip personally identifiable information from the customer's input, you can use the postmessage webhook to add it back. Or if you use the premessage webhook to translate the customer's input to the language of the assistant, you can use the postmessage webhook to translate the response back into the customer's native language before it is returned. For more information, see [Making a call before processing a message](#).

 **Note:** For environments where private endpoints are in use, keep in mind that a webhook sends traffic over the internet.

## Defining the webhook

You can define one webhook URL to use for processing every message response before it is sent to the channel and shown to

the customer.

The programmatic call to the external service must meet these requirements:

- The call must be a POST HTTP request.
- The call must be completed in 30 seconds or less.
- The format of the request and response must be in JSON. For example:Content-Type: application/json.

**⚠ Important:** Do not set up and test your webhook in a production environment where the assistant is deployed and is interacting with customers.

To add the webhook details, complete the following steps:

1. In your assistant, open the environment where you want to configure the webhook.
2. Click the  icon to open the environment settings.
3. On the **Environment settings** page, click **Webhooks > Post-message webhook**.
4. Set the *Post-message webhook* switch to **Enabled**.
5. Decide whether to return an error if the webhook call fails.

**⚠ Important:** When enabled, everything stops until the processing step is completed successfully.

- If you have a critical postprocessing step that must be taken before you want to allow the response to be sent to the customer, then enable this setting.
- When this setting is disabled, the assistant ignores any errors it encounters and continues to process the response without taking the processing step. If the postprocessing step is helpful but not critical, consider keeping this setting disabled.

6. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

For example, maybe you store your assistant's responses in a separate content management system. When the assistant understands the input, the processed action returns a unique ID that corresponds to a response in your CMS. To call a service that retrieves a response from your CMS for a given unique ID, specify the URL for your service instance (for example, `https://example.com/get_answer`).

You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.

**⚠ Important:** You cannot use a webhook to call a Cloud Functions action that uses token-based Identity and Access Management (IAM) authentication. However, you can make a call to a Cloud Functions web action or a secured web action.

7. In the **Secret** field, add a private key to pass with the request that can be used to authenticate with the external service.

The key must be specified as a text string, such as `purple unicorn`. The maximum length is 1,024 characters. You cannot specify a context variable.

It is the responsibility of the external service to check for and verify the secret. If the external service does not require a token, specify any string you want. You cannot leave this field empty.

8. In the **Timeout** field, specify the length of time (in seconds) you want the assistant to wait for a response from the webhook before returning an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.

9. In the Headers section, add any headers that you want to pass to the service one at a time by clicking **Add header**.

For example, if the external application that you call returns a response, it might be able to send a response in multiple different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header that indicates that you want the resulting value to be returned in JSON format.

Header name	Header value
Content-Type	application/json

#### Header example

The service automatically sends an `Authorization` header with a JWT; you do not need to add one. If you want to handle authorization yourself, add your own authorization header and it will be used instead.

Your webhook details are saved automatically.

## Testing the webhook

 **Important:** Do extensive testing of your webhook before you enable it for an assistant that is being used in a production environment.

The webhook is triggered only when a message has been processed by your assistant, and a response is ready to be returned to the channel.

If you enable the setting that returns an error when the webhook call fails, the processing of the assistant is halted entirely if the webhook encounters any issues. Take steps to test the process that you are calling on a regular basis so you will be alerted if the external service is down, and can take actions to prevent all of the message responses from failing to be returned.

**Tip:** If you call an Cloud Functions web action, you can use the logging capability in Cloud Functions to help you troubleshoot your code. You can [download the command line interface](#), and then enable logging with the [activation polling command](#).

## Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you will know that your webhook has an issue if every test message you submit returns a message such as, `There is an error with the message you just sent, but feel free to ask me something else.` If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Webhook responded with [500] status code	There's a problem with the external service you called. The code failed or the external server refused the request.

500 Processor Exception :[connections to all backends failing]

An error occurred in the webhook microservice. It could not connect to backend services.

#### Error code details

## Webhook security

To authenticate the webhook request, verify the JSON Web Token (JWT) that is sent with the request. The webhook microservice automatically generates a JWT and sends it in the `Authorization` header with each webhook call. It is your responsibility to add code to the external service that verifies the JWT.

For example, if you specify `purple unicorn` in the **Secret** field, you might add code similar to this:

```
const jwt = require('jsonwebtoken');
...
const token = request.headers.authentication; // grab the "Authentication" header
try {
  const decoded = jwt.verify(token, 'purple unicorn');
} catch(err) {
  // error thrown if token is invalid
}
```

## Example request body

It is useful to know the format of the request postmessage webhook body so that your external code can process it.

The payload contains the response body that is returned by your assistant for the `v2 /message` (stateful and stateless) API call. The event name `message_processed` indicates that the request is generated by the postmessage webhook. For more information about the message request body, see the [API reference](#).

The following sample shows how a simple request body is formatted.

```
{
  "event": {
    "name": "message_processed"
  },
  "options": {},
  "payload": {
    "output": {
      "intents": [
        {
          "intent": "General_Greetings",
          "confidence": 1
        }
      ],
      "entities": [],
      "generic": [
        {
          "response_type": "text",
          "text": "Hello. Good evening"
        }
      ]
    },
    "user_id": "test user",
    "context": {
      "global": {
        "system": {
          "user_id": "test user",
          "turn_count": 11
        },
        "session_id": "sxxx"
      }
    }
}
```

```
        "skills": {
            "actions_skill": {
                "user_defined": {
                    "var": "anthony"
                },
                "system": {
                    "state": "nnn"
                }
            }
        }
    }
}
```

### Example 1

This example shows you how to add `y'all` to the end of each response from the assistant.

In the postmessage webhook configuration page, the following values are specified:

- **URL:** `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/southernize/add_southern_charm`
  - **Secret:** none
  - **Header name:** Content-Type
  - **Header value:** application/json

The postmessage webhook calls an IBM Cloud Functions web action name `add_southern_charm`.

The node.js code in the `add_southern_charm` web action looks as follows:

### Example 2

This example shows you how to translate a message response back to the customer's native language. It only works if you

perform the steps in [Example 2](#) to define a premessage webhook that translates the original message into English.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks for the language of the original incoming text, which you stored in a context variable named `original_input` in the premessage webhook code. The second action in the sequence translates the dialog response text from English into the original language that was used by the customer.

In the premessage webhook configuration page, the following values are specified:

- **URL:** `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/response-translation_sequence`
- **Secret:** none
- **Header name:** Content-Type
- **Header value:** application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
  console.log(JSON.stringify(params))

  if (params.payload.output.generic[0].text !== '') {
    const options = { method: 'POST',
      url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
      auth: {
        'username': 'apikey',
        'password': 'nnnn'
      },
      headers: {
        "Content-Type": "text/plain"
      },
      body: [
        params.payload.context.skills['actions skill'].user_defined.original_input
      ],
      json: true,
    };
    return rp(options)
      .then(res => {
        //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
        params.payload.context.skills['actions skill'].user_defined['language'] =
        res.languages[0].language;
        console.log(JSON.stringify(params))
        return params;
      })
    }
  else {
    params.payload.context.skills['actions skill'].user_defined['language'] = 'none';
    return params;
  }
};
```

The second web action in the sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
  console.log(JSON.stringify(params))
  if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
    const options = { method: 'POST',
      url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-
```

```

b693-3bc63869nnnn/v3/translate?version=2018-05-01',
  auth: {
    'username': 'apikey',
    'password': 'nnn'
  },
  body: {
    text: [
      params.payload.output.generic[0].text
    ],
    target: params.payload.context.skills["actions skill"].user_defined.language
  },
  json: true
};

  return rp(options)
  .then(res => {
    params.payload.context.skills["actions skill"].user_defined["original_output"] =
    params.payload.output.generic[0].text;
    params.payload.output.generic[0].text = res.translations[0].translation;
    return {
      body : params
    }
  })
}

return {
  body : params
}
};

```

## Removing the webhook

If you decide you do not want to process message responses with a webhook, complete the following steps:

1. From the assistant overview page, click the  icon, and then choose **Settings**.
2. Click **Webhooks > Post-message webhook**.
3. Do one of the following things:
  - To change the webhook that you want to call, click **Delete webhook** to delete the currently specified URL and secret. You can then add a new URL and other details.
  - To stop calling a webhook to process every message response, click the *Post-message webhook* switch to disable the webhook altogether.

## Logging activity with a webhook

You can log activity by making a call to an external service or application every time a customer submits input to the assistant.

A webhook is a mechanism that allows you to call out to an external program based on events in your program.

This feature is available only to Plus and Enterprise plan users.



**Note:** The Plus plan allows no more than 5 log webhooks per instance. This limit does not apply to Enterprise plan instances.

Add a log webhook to your assistant if you want to use an external service to log Watson Assistant activity. You can log two kinds of activity:

- **Messages and responses:** The log webhook is triggered each time the assistant responds to customer input. You can use this option as an alternative to the built-in analytics feature to handle logging yourself. (For more information about the built-in analytics support, see [Review your entire assistant at a glance](#).)

**⚠ Important:** If you are using a custom channel, note that the log webhook works with the v2 `/message` API only (stateless and stateful). For more information, see the [API reference](#). All built-in channel integrations use this API.

- **Call detail records (CDRs):** The log webhook is triggered after each telephone call a user makes to your assistant using the phone integration. A Call Detail Record (CDR) is a summary report that documents the details of a telephone call, including phone numbers, call length, latency, and other diagnostic information. CDR records are only for assistants that use the phone integration.

The log webhook does not return anything to your assistant.



**Note:** For environments where private endpoints are in use, keep in mind that a webhook sends traffic over the internet.

## Defining the webhook

You can define one webhook URL to use for logging every incoming message or CDR event.

The programmatic call to the external service must meet these requirements:

- The call must be a POST HTTP request.

To add the webhook details, complete the following steps:

1. In your assistant, go to **Environments** and open the environment where you want to configure the webhook.
2. On the **Environments** page, click the icon beside the environment title to open the environment settings.
3. On the **Environment settings** page, click **Webhooks > Log webhook**.
4. Set the *Log webhook* switch to **Enabled**.

If you cannot enable the webhook, you might need to upgrade your service plan.

5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts (for example, `https://example.com/my_log_service`).

You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.

6. In the **Secret** field, add a token to pass with the request that can be used to authenticate with the external service.

The secret must be specified as a text string, such as `purple_unicorn`. The maximum length is 1,024 characters. You cannot specify a context variable.

It is the responsibility of the external service to check for and verify the secret. If the external service does not require a token, specify any string you want. You cannot leave this field empty.

7. Click the appropriate checkboxes to select which kinds of activity you want to log:

- To log messages and responses, select **Subscribe to conversation logs**.
- To log CDR events for the phone integration, select **Subscribe to CDR (Call Detail Records)**.

8. In the Headers section, add any headers that you want to pass to the service one at a time by clicking **Add header**.

The service automatically sends an `Authorization` header with a JWT; you do not need to add one. If you want to handle authorization yourself, add your own authorization header and it will be used instead.

Your webhook details are saved automatically.

## Removing the webhook

If you decide you do not want to log messages with a webhook, complete the following steps:

1. In your assistant, go to **Environments** and open the environment where you want to remove the webhook.
2. On the **Environments** page, click the  icon beside the environment title to open the environment settings.
3. On the **Environment settings** page, click **Webhooks > Log webhook**.
4. Do one of the following things:
  - To change the webhook that you want to call, click **Delete webhook** to delete the currently specified URL and secret. You can then add a new URL and other details.
  - To stop calling a webhook to log every message and response, click the **Log webhook** switch to disable the webhook altogether.

## Webhook security

To authenticate the webhook request, verify the JSON Web Token (JWT) that is sent with the request. The webhook microservice automatically generates a JWT and sends it in the `Authorization` header with each webhook call. It is your responsibility to add code to the external service that verifies the JWT.

For example, if you specify `purple unicorn` in the **Secret** field, you might add code similar to this:

```
const jwt = require('jsonwebtoken');
...
const token = request.headers.authentication; // grab the "Authentication" header
try {
  const decoded = jwt.verify(token, 'purple unicorn');
} catch(err) {
  // error thrown if token is invalid
}
```

## Webhook request body

The request body the webhook sends to the external service is a JSON object with the following structure:

```
{
  "event": {
    "name": "{event_type}"
  },
  "payload": {
    ...
  }
}
```

where `{event_type}` is either `message_logged` (for messages and responses) or `cdr_logged` (for CDR events).

The `payload` object contains the event data to be logged. The structure of the `payload` object depends on the type of event.

### Message event payload

For `message_logged` events, the `payload` object contains data about a message request sent to the assistant and the message response returned to the integration or client application. For more information about the fields that are part of message requests and responses, see the [API reference](#).

 **Important:** The log webhook payload might include data that is not currently supported by the API. Any fields that are not defined in the API reference documentation are subject to change.

## CDR event payload

For `cdr_logged` events, the `payload` object contains data about a Call Detail Record (CDR) event that was handled by the phone integration. The structure of the `payload` object for a CDR event is as shown by this example:

```
{  
  "primary_phone_number": "+18005550123",  
  "global_session_id": "9caa8bad-aaa8-4a5a-a4b5-62bcc703d15",  
  "failure_occurred": false,  
  "transfer_occurred": false,  
  "active_calls": 0,  
  "warnings_and_errors": [  
    {  
      "code": "CWSMR0033W",  
      "message": "CWSMR0033W: The inbound RTP audio stream jitter of 43 ms exceeds the maximum jitter threshold of 30 ms."  
    },  
    {  
      "code": "CWSMR0070W",  
      "message": "CWSMR0070W: A request to the Watson Speech To Text service failed for the following reason = Unexpected server response: 403, response headers = {\\"strict-transport-security\\":\\"max-age=31536000; includeSubDomains;\\",\\"content-length\\":\\"157\\",\\"content-type\\":\\"application/json\\",\\"x-dp-watson-tran-id\\":\\"23860083-88b6-41d7-9130-30bbfebe647e\\",\\"x-request-id\\":\\"23860083-88b6-41d7-9130-30bbfebe647e\\",\\"x-global-transaction-id\\":\\"6c764df3-81db-41bb-a14f-62384facffca\\",\\"server\\":\\"watson-gateway\\",\\"x-edgeconnect-midmile-rtt\\":\\"1\\",\\"x-edgeconnect-origin-mex-latency\\":\\"28\\",\\"date\\":\\"Thu, 13 May 2021 20:31:12 GMT\\",\\"connection\\":\\"keep-alive\\", response body = {\\"code\\":403,\\\"trace\\":\\"23860083-88b6-41d7-9130-30bbfebe647e\\",\\\"error\\":\\"Forbidden\\",\\\"more_info\\":\\"[https://cloud.ibm.com/docs/watson?topic=watson-forbidden-error](https://cloud.ibm.com/docs/watson?topic=watson-forbidden-error)\\",\\\"x-global-transaction-id\\": 6c764df3-81db-41bb-a14f-62384facffca. The Media Relay will reattempt to send the request."  
    }  
  ],  
  "realtime_transport_network_summary": {  
    "inbound_stream": {  
      "average_jitter": 4,  
      "canonical_name": "b74f3689-1ae8-4a0a-bde3-adf5b488553e",  
      "maximum_jitter": 18,  
      "packets_lost": 0,  
      "packets_transmitted": 952,  
      "tool_name": ""  
    },  
    "outbound_stream": {  
      "average_jitter": 0,  
      "canonical_name": "voice.gateway",  
      "maximum_jitter": 0,  
      "packets_lost": 0,  
      "packets_transmitted": 838,  
      "tool_name": "IBM Voice Gateway/1.0.7.0"  
    }  
  },  
  "call": {  
    "start_timestamp": "2021-10-12T20:54:02.591Z",  
    "stop_timestamp": "2021-10-12T20:54:20.375Z",  
    "milliseconds_elapsed": 17784,  
    "outbound": false,  
    "end_reason": "assistant_hangup",  
    "security": {  
      "media_encrypted": false,  
      "signaling_encrypted": false,  
      "sip_authenticated": false  
    }  
  },  
  "session_initiation_protocol": {  
    "invite_arrival_time": "2021-10-12T20:54:00.565Z",  
    "setup_milliseconds": 2026,  
    "headers": {  
      "call_id": "17465345_115257202@10.90.150.99",  
      "correlation_id": "17465345_115257202@10.90.150.99",  
      "method": "INVITE",  
      "uri": "sip:17465345@10.90.150.99",  
      "version": "1.0",  
      "content_type": "application/sdp",  
      "content_length": "1024",  
      "content": "v=0;avp=1;rpid=17465345;avp=2;rpid=17465345;avp=3;rpid=17465345;avp=4;rpid=17465345;avp=5;rpid=17465345;avp=6;rpid=17465345;avp=7;rpid=17465345;avp=8;rpid=17465345;avp=9;rpid=17465345;avp=10;rpid=17465345;avp=11;rpid=17465345;avp=12;rpid=17465345;avp=13;rpid=17465345;avp=14;rpid=17465345;avp=15;rpid=17465345;avp=16;rpid=17465345;avp=17;rpid=17465345;avp=18;rpid=17465345;avp=19;rpid=17465345;avp=20;rpid=17465345;avp=21;rpid=17465345;avp=22;rpid=17465345;avp=23;rpid=17465345;avp=24;rpid=17465345;avp=25;rpid=17465345;avp=26;rpid=17465345;avp=27;rpid=17465345;avp=28;rpid=17465345;avp=29;rpid=17465345;avp=30;rpid=17465345;avp=31;rpid=17465345;avp=32;rpid=17465345;avp=33;rpid=17465345;avp=34;rpid=17465345;avp=35;rpid=17465345;avp=36;rpid=17465345;avp=37;rpid=17465345;avp=38;rpid=17465345;avp=39;rpid=17465345;avp=40;rpid=17465345;avp=41;rpid=17465345;avp=42;rpid=17465345;avp=43;rpid=17465345;avp=44;rpid=17465345;avp=45;rpid=17465345;avp=46;rpid=17465345;avp=47;rpid=17465345;avp=48;rpid=17465345;avp=49;rpid=17465345;avp=50;rpid=17465345;avp=51;rpid=17465345;avp=52;rpid=17465345;avp=53;rpid=17465345;avp=54;rpid=17465345;avp=55;rpid=17465345;avp=56;rpid=17465345;avp=57;rpid=17465345;avp=58;rpid=17465345;avp=59;rpid=17465345;avp=60;rpid=17465345;avp=61;rpid=17465345;avp=62;rpid=17465345;avp=63;rpid=17465345;avp=64;rpid=17465345;avp=65;rpid=17465345;avp=66;rpid=17465345;avp=67;rpid=17465345;avp=68;rpid=17465345;avp=69;rpid=17465345;avp=70;rpid=17465345;avp=71;rpid=17465345;avp=72;rpid=17465345;avp=73;rpid=17465345;avp=74;rpid=17465345;avp=75;rpid=17465345;avp=76;rpid=17465345;avp=77;rpid=17465345;avp=78;rpid=17465345;avp=79;rpid=17465345;avp=80;rpid=17465345;avp=81;rpid=17465345;avp=82;rpid=17465345;avp=83;rpid=17465345;avp=84;rpid=17465345;avp=85;rpid=17465345;avp=86;rpid=17465345;avp=87;rpid=17465345;avp=88;rpid=17465345;avp=89;rpid=17465345;avp=90;rpid=17465345;avp=91;rpid=17465345;avp=92;rpid=17465345;avp=93;rpid=17465345;avp=94;rpid=17465345;avp=95;rpid=17465345;avp=96;rpid=17465345;avp=97;rpid=17465345;avp=98;rpid=17465345;avp=99;rpid=17465345;avp=100;rpid=17465345;avp=101;rpid=17465345;avp=102;rpid=17465345;avp=103;rpid=17465345;avp=104;rpid=17465345;avp=105;rpid=17465345;avp=106;rpid=17465345;avp=107;rpid=17465345;avp=108;rpid=17465345;avp=109;rpid=17465345;avp=110;rpid=17465345;avp=111;rpid=17465345;avp=112;rpid=17465345;avp=113;rpid=17465345;avp=114;rpid=17465345;avp=115;rpid=17465345;avp=116;rpid=17465345;avp=117;rpid=17465345;avp=118;rpid=17465345;avp=119;rpid=17465345;avp=120;rpid=17465345;avp=121;rpid=17465345;avp=122;rpid=17465345;avp=123;rpid=17465345;avp=124;rpid=17465345;avp=125;rpid=17465345;avp=126;rpid=17465345;avp=127;rpid=17465345;avp=128;rpid=17465345;avp=129;rpid=17465345;avp=130;rpid=17465345;avp=131;rpid=17465345;avp=132;rpid=17465345;avp=133;rpid=17465345;avp=134;rpid=17465345;avp=135;rpid=17465345;avp=136;rpid=17465345;avp=137;rpid=17465345;avp=138;rpid=17465345;avp=139;rpid=17465345;avp=140;rpid=17465345;avp=141;rpid=17465345;avp=142;rpid=17465345;avp=143;rpid=17465345;avp=144;rpid=17465345;avp=145;rpid=17465345;avp=146;rpid=17465345;avp=147;rpid=17465345;avp=148;rpid=17465345;avp=149;rpid=17465345;avp=150;rpid=17465345;avp=151;rpid=17465345;avp=152;rpid=17465345;avp=153;rpid=17465345;avp=154;rpid=17465345;avp=155;rpid=17465345;avp=156;rpid=17465345;avp=157;rpid=17465345;avp=158;rpid=17465345;avp=159;rpid=17465345;avp=160;rpid=17465345;avp=161;rpid=17465345;avp=162;rpid=17465345;avp=163;rpid=17465345;avp=164;rpid=17465345;avp=165;rpid=17465345;avp=166;rpid=17465345;avp=167;rpid=17465345;avp=168;rpid=17465345;avp=169;rpid=17465345;avp=170;rpid=17465345;avp=171;rpid=17465345;avp=172;rpid=17465345;avp=173;rpid=17465345;avp=174;rpid=17465345;avp=175;rpid=17465345;avp=176;rpid=17465345;avp=177;rpid=17465345;avp=178;rpid=17465345;avp=179;rpid=17465345;avp=180;rpid=17465345;avp=181;rpid=17465345;avp=182;rpid=17465345;avp=183;rpid=17465345;avp=184;rpid=17465345;avp=185;rpid=17465345;avp=186;rpid=17465345;avp=187;rpid=17465345;avp=188;rpid=17465345;avp=189;rpid=17465345;avp=190;rpid=17465345;avp=191;rpid=17465345;avp=192;rpid=17465345;avp=193;rpid=17465345;avp=194;rpid=17465345;avp=195;rpid=17465345;avp=196;rpid=17465345;avp=197;rpid=17465345;avp=198;rpid=17465345;avp=199;rpid=17465345;avp=200;rpid=17465345;avp=201;rpid=17465345;avp=202;rpid=17465345;avp=203;rpid=17465345;avp=204;rpid=17465345;avp=205;rpid=17465345;avp=206;rpid=17465345;avp=207;rpid=17465345;avp=208;rpid=17465345;avp=209;rpid=17465345;avp=210;rpid=17465345;avp=211;rpid=17465345;avp=212;rpid=17465345;avp=213;rpid=17465345;avp=214;rpid=17465345;avp=215;rpid=17465345;avp=216;rpid=17465345;avp=217;rpid=17465345;avp=218;rpid=17465345;avp=219;rpid=17465345;avp=220;rpid=17465345;avp=221;rpid=17465345;avp=222;rpid=17465345;avp=223;rpid=17465345;avp=224;rpid=17465345;avp=225;rpid=17465345;avp=226;rpid=17465345;avp=227;rpid=17465345;avp=228;rpid=17465345;avp=229;rpid=17465345;avp=230;rpid=17465345;avp=231;rpid=17465345;avp=232;rpid=17465345;avp=233;rpid=17465345;avp=234;rpid=17465345;avp=235;rpid=17465345;avp=236;rpid=17465345;avp=237;rpid=17465345;avp=238;rpid=17465345;avp=239;rpid=17465345;avp=240;rpid=17465345;avp=241;rpid=17465345;avp=242;rpid=17465345;avp=243;rpid=17465345;avp=244;rpid=17465345;avp=245;rpid=17465345;avp=246;rpid=17465345;avp=247;rpid=17465345;avp=248;rpid=17465345;avp=249;rpid=17465345;avp=250;rpid=17465345;avp=251;rpid=17465345;avp=252;rpid=17465345;avp=253;rpid=17465345;avp=254;rpid=17465345;avp=255;rpid=17465345;avp=256;rpid=17465345;avp=257;rpid=17465345;avp=258;rpid=17465345;avp=259;rpid=17465345;avp=260;rpid=17465345;avp=261;rpid=17465345;avp=262;rpid=17465345;avp=263;rpid=17465345;avp=264;rpid=17465345;avp=265;rpid=17465345;avp=266;rpid=17465345;avp=267;rpid=17465345;avp=268;rpid=17465345;avp=269;rpid=17465345;avp=270;rpid=17465345;avp=271;rpid=17465345;avp=272;rpid=17465345;avp=273;rpid=17465345;avp=274;rpid=17465345;avp=275;rpid=17465345;avp=276;rpid=17465345;avp=277;rpid=17465345;avp=278;rpid=17465345;avp=279;rpid=17465345;avp=280;rpid=17465345;avp=281;rpid=17465345;avp=282;rpid=17465345;avp=283;rpid=17465345;avp=284;rpid=17465345;avp=285;rpid=17465345;avp=286;rpid=17465345;avp=287;rpid=17465345;avp=288;rpid=17465345;avp=289;rpid=17465345;avp=290;rpid=17465345;avp=291;rpid=17465345;avp=292;rpid=17465345;avp=293;rpid=17465345;avp=294;rpid=17465345;avp=295;rpid=17465345;avp=296;rpid=17465345;avp=297;rpid=17465345;avp=298;rpid=17465345;avp=299;rpid=17465345;avp=300;rpid=17465345;avp=301;rpid=17465345;avp=302;rpid=17465345;avp=303;rpid=17465345;avp=304;rpid=17465345;avp=305;rpid=17465345;avp=306;rpid=17465345;avp=307;rpid=17465345;avp=308;rpid=17465345;avp=309;rpid=17465345;avp=310;rpid=17465345;avp=311;rpid=17465345;avp=312;rpid=17465345;avp=313;rpid=17465345;avp=314;rpid=17465345;avp=315;rpid=17465345;avp=316;rpid=17465345;avp=317;rpid=17465345;avp=318;rpid=17465345;avp=319;rpid=17465345;avp=320;rpid=17465345;avp=321;rpid=17465345;avp=322;rpid=17465345;avp=323;rpid=17465345;avp=324;rpid=17465345;avp=325;rpid=17465345;avp=326;rpid=17465345;avp=327;rpid=17465345;avp=328;rpid=17465345;avp=329;rpid=17465345;avp=330;rpid=17465345;avp=331;rpid=17465345;avp=332;rpid=17465345;avp=333;rpid=17465345;avp=334;rpid=17465345;avp=335;rpid=17465345;avp=336;rpid=17465345;avp=337;rpid=17465345;avp=338;rpid=17465345;avp=339;rpid=17465345;avp=340;rpid=17465345;avp=341;rpid=17465345;avp=342;rpid=17465345;avp=343;rpid=17465345;avp=344;rpid=17465345;avp=345;rpid=17465345;avp=346;rpid=17465345;avp=347;rpid=17465345;avp=348;rpid=17465345;avp=349;rpid=17465345;avp=350;rpid=17465345;avp=351;rpid=17465345;avp=352;rpid=17465345;avp=353;rpid=17465345;avp=354;rpid=17465345;avp=355;rpid=17465345;avp=356;rpid=17465345;avp=357;rpid=17465345;avp=358;rpid=17465345;avp=359;rpid=17465345;avp=360;rpid=17465345;avp=361;rpid=17465345;avp=362;rpid=17465345;avp=363;rpid=17465345;avp=364;rpid=17465345;avp=365;rpid=17465345;avp=366;rpid=17465345;avp=367;rpid=17465345;avp=368;rpid=17465345;avp=369;rpid=17465345;avp=370;rpid=17465345;avp=371;rpid=17465345;avp=372;rpid=17465345;avp=373;rpid=17465345;avp=374;rpid=17465345;avp=375;rpid=17465345;avp=376;rpid=17465345;avp=377;rpid=17465345;avp=378;rpid=17465345;avp=379;rpid=17465345;avp=380;rpid=17465345;avp=381;rpid=17465345;avp=382;rpid=17465345;avp=383;rpid=17465345;avp=384;rpid=17465345;avp=385;rpid=17465345;avp=386;rpid=17465345;avp=387;rpid=17465345;avp=388;rpid=17465345;avp=389;rpid=17465345;avp=390;rpid=17465345;avp=391;rpid=17465345;avp=392;rpid=17465345;avp=393;rpid=17465345;avp=394;rpid=17465345;avp=395;rpid=17465345;avp=396;rpid=17465345;avp=397;rpid=17465345;avp=398;rpid=17465345;avp=399;rpid=17465345;avp=400;rpid=17465345;avp=401;rpid=17465345;avp=402;rpid=17465345;avp=403;rpid=17465345;avp=404;rpid=17465345;avp=405;rpid=17465345;avp=406;rpid=17465345;avp=407;rpid=17465345;avp=408;rpid=17465345;avp=409;rpid=17465345;avp=410;rpid=17465345;avp=411;rpid=17465345;avp=412;rpid=17465345;avp=413;rpid=17465345;avp=414;rpid=17465345;avp=415;rpid=17465345;avp=416;rpid=17465345;avp=417;rpid=17465345;avp=418;rpid=17465345;avp=419;rpid=17465345;avp=420;rpid=17465345;avp=421;rpid=17465345;avp=422;rpid=17465345;avp=423;rpid=17465345;avp=424;rpid=17465345;avp=425;rpid=17465345;avp=426;rpid=17465345;avp=427;rpid=17465345;avp=428;rpid=17465345;avp=429;rpid=17465345;avp=430;rpid=17465345;avp=431;rpid=17465345;avp=432;rpid=17465345;avp=433;rpid=17465345;avp=434;rpid=17465345;avp=435;rpid=17465345;avp=436;rpid=17465345;avp=437;rpid=17465345;avp=438;rpid=17465345;avp=439;rpid=17465345;avp=440;rpid=17465345;avp=441;rpid=17465345;avp=442;rpid=17465345;avp=443;rpid=17465345;avp=444;rpid=17465345;avp=445;rpid=17465345;avp=446;rpid=17465345;avp=447;rpid=17465345;avp=448;rpid=17465345;avp=449;rpid=17465345;avp=450;rpid=17465345;avp=451;rpid=17465345;avp=452;rpid=17465345;avp=453;rpid=17465345;avp=454;rpid=17465345;avp=455;rpid=17465345;avp=456;rpid=17465345;avp=457;rpid=17465345;avp=458;rpid=17465345;avp=459;rpid=17465345;avp=460;rpid=17465345;avp=461;rpid=17465345;avp=462;rpid=17465345;avp=463;rpid=17465345;avp=464;rpid=17465345;avp=465;rpid=17465345;avp=466;rpid=17465345;avp=467;rpid=17465345;avp=468;rpid=17465345;avp=469;rpid=17465345;avp=470;rpid=17465345;avp=471;rpid=17465345;avp=472;rpid=17465345;avp=473;rpid=17465345;avp=474;rpid=17465345;avp=475;rpid=17465345;avp=476;rpid=17465345;avp=477;rpid=17465345;avp=478;rpid=17465345;avp=479;rpid=17465345;avp=480;rpid=17465345;avp=481;rpid=17465345;avp=482;rpid=17465345;avp=483;rpid=17465345;avp=484;rpid=17465345;avp=485;rpid=17465345;avp=486;rpid=17465345;avp=487;rpid=17465345;avp=488;rpid=17465345;avp=489;rpid=17465345;avp=490;rpid=17465345;avp=491;rpid=17465345;avp=492;rpid=17465345;avp=493;rpid=17465345;avp=494;rpid=17465345;avp=495;rpid=17465345;avp=496;rpid=17465345;avp=497;rpid=17465345;avp=498;rpid=17465345;avp=499;rpid=17465345;avp=500;rpid=17465345;avp=501;rpid=17465345;avp=502;rpid=17465345;avp=503;rpid=17465345;avp=504;rpid=17465345;avp=505;rpid=17465345;avp=506;rpid=17465345;avp=507;rpid=17465345;avp=508;rpid=17465345;avp=509;rpid=17465345;avp=510;rpid=17465345;avp=511;rpid=17465345;avp=512;rpid=17465345;avp=513;rpid=17465345;avp=514;rpid=17465345;avp=515;rpid=17465345;avp=516;rpid=17465345;avp=517;rpid=17465345;avp=518;rpid=17465345;avp=519;rpid=17465345;avp=520;rpid=17465345;avp=521;rpid=17465345;avp=522;rpid=17465345;avp=523;rpid=17465345;avp=524;rpid=17465345;avp=525;rpid=17465345;avp=526;rpid=17465345;avp=527;rpid=17465345;avp=528;rpid=17465345;avp=529;rpid=17465345;avp=530;rpid=17465345;avp=531;rpid=17465345;avp=532;rpid=17465345;avp=533;rpid=17465345;avp=534;rpid=17465345;avp=535;rpid=17465345;avp=536;rpid=17465345;avp=537;rpid=17465345;avp=538;rpid=17465345;avp=539;rpid=17465345;avp=540;rpid=17465345;avp=541;rpid=17465345;avp=542;rpid=17465345;avp=543;rpid=17465345;avp=544;rpid=17465345;avp=545;rpid=17465345;avp=546;rpid=17465345;avp=547;rpid=17465345;avp=548;rpid=17465345;avp=549;rpid=17465345;avp=550;rpid=17465345;avp=551;rpid=17465345;avp=552;rpid=17465345;avp=553;rpid=17465345;avp=554;rpid=17465345;avp=555;rpid=17465345;avp=556;rpid=17465345;avp=557;rpid=17465345;avp=558;rpid=17465345;avp=559;rpid=174
```

```

    "from_uri": "sip:+18885550456@pstn.twilio.com",
    "to_uri": "sip:+18005550123@public.voip.us-south.assistant.test.watson.cloud.ibm.com"
  },
  "max_response_milliseconds": {
    "assistant": 339,
    "text_to_speech": 535,
    "speech_to_text": 0
  },
  "assistant_interaction_summaries": [
    {
      "session_id": "7874ec3a-1330-4180-afe1-46bfb220af5b",
      "assistant_id": "97f16ba4-ad94-41af-aa6c-33cd56ad5e7e",
      "turns": [
        {
          "assistant": {
            "log_id": "58bebfd1-0118-419b-a555-b152a1efbbe8",
            "response_milliseconds": 339,
            "start_timestamp": "2021-10-12T20:54:00.722Z"
          },
          "request": {
            "type": "start"
          },
          "response": [
            {
              "barge_in_occurred": true,
              "streaming_statistics": {
                "response_milliseconds": 301,
                "start_timestamp": "2021-10-12T20:54:00.722Z",
                "stop_timestamp": "2021-10-12T20:54:01.023Z",
                "transaction_id": "3dce431c-fb2f-4b62-9fce-585f4e06fe00"
              },
              "type": "text_to_speech"
            }
          ]
        },
        {
          "assistant": {
            "log_id": "38f36bfb-c2aa-4600-9418-6ab422664e31",
            "response_milliseconds": 158,
            "start_timestamp": "2021-10-12T20:54:05.621Z"
          },
          "request": {
            "type": "dtmf"
          },
          "response": [
            {
              "type": "disable_speech_barge_in"
            },
            {
              "type": "text_to_speech",
              "barge_in_occurred": false,
              "streaming_statistics": {
                "transaction_id": "af4c47c3-5cc4-43c8-9b9c-81d6f997c52f",
                "start_timestamp": "2021-10-12T20:54:06.321Z",
                "stop_timestamp": "2021-10-12T20:54:14.338Z",
                "response_milliseconds": 535
              }
            },
            {
              "type": "enable_speech_barge_in"
            },
            {
              "type": "text_to_speech",
              "barge_in_occurred": true,
              "streaming_statistics": {
                "transaction_id": "eafdd846-2829-4e1a-8068-b1035510b1e1",
                "start_timestamp": "2021-10-12T20:54:14.795Z",
                "stop_timestamp": "2021-10-12T20:54:20.388Z",
                "response_milliseconds": 535
              }
            }
          ]
        }
      ]
    }
  ]
}

```

```

        "response_milliseconds": 447
    }
}
],
{
    "assistant": {
        "log_id": "07d74b35-0205-43e4-923c-1e43e1cb429c",
        "response_milliseconds": 0,
        "start_timestamp": "2021-10-12T20:54:20.377Z"
    },
    "request": {
        "type": "hangup"
    },
    "response": []
}
]
}
}

```

For detailed information about the structure of the CDR event payload, see [CDR log event reference](#).

## Building a custom extension

If you need to integrate your assistant with an external service that has a REST API, you can build a custom extension by importing an OpenAPI document.

After you create a custom extension, you can connect it to an assistant as an integration. In your actions, you can then define steps that interact with the external service by calling the extension.

**Tip:** The [Watson Assistant extension starter kit](#) repo on GitHub provides files you can use to quickly build a working extension. Each starter kit includes a tested OpenAPI definition you can use to create an extension that accesses a third-party API, along with a downloadable JSON file you can import to create an assistant that accesses the extension.

## Overview

OpenAPI (formerly known as Swagger) is an open standard for describing and documenting REST APIs. An OpenAPI document defines the resources and operations supported by an API, including request parameters and response data, along with details such as server URLs and authentication methods.

An OpenAPI document describes a REST API in terms of *paths* and *operations*. A path identifies a particular resource that can be accessed using the API (for example, a hotel reservation or a customer record), while an *operation* defines a particular action that can be performed on that resource (such as creating, retrieving, updating, or deleting it). The OpenAPI document specifies all of the details for each operation, including the HTTP method used, request parameters, the data included in the request body, and the structure of the response body.

For more information about the OpenAPI specification, see [OpenAPI Specification](#).

When you create a custom extension, you import an OpenAPI document that describes the REST API of an external service. Watson Assistant parses the OpenAPI document to identify the operations supported by the external service, along with information about the input parameters and response for each operation and supported authentication methods.

After this processing has completed, the custom extension becomes available as a new integration that you can connect to the assistant. Your assistant can then use the extension to send requests to the external service based on conversations with your customers. Values included in the response from the service are then mapped to action variables, which can be accessed by

subsequent action steps.

(For more information about connecting a custom extension to an assistant, see [Add a custom extension](#).)

## Preparing the API definition

To create a custom extension, you need access to an OpenAPI document that describes the REST API you want to integrate with. Many third-party services publish OpenAPI documents that describe their APIs, which you can download and import. For an API that your company maintains, you can use standard tools to create an OpenAPI document describing it.

**Tip:** The [SwaggerHub](#) website offers an [OpenAPI 3.0 Tutorial](#), as well as [tools](#) to help you develop and validate your OpenAPI document. You can use the online [Swagger editor](#) to convert your OpenAPI document to the correct format and OpenAPI version.

The OpenAPI document must satisfy the following requirements and restrictions:

- The document must conform to the OpenAPI 3.0 specification. If you have an OpenAPI (or Swagger) document that uses an earlier version of the specification, you can use the online [Swagger editor](#) to convert it to OpenAPI 3.0.
- The document must be in JSON format (YAML is not supported). If you have a YAML document, you can use the online [Swagger editor](#) to convert it to JSON.
- Each operation must have a clear and concise `summary`. The text of the summary is used in the UI to describe the operations that are available from an action, so it should be short and meaningful to someone who is building an assistant.
- [Relative URLs](#) are currently not supported.
- Only `Basic`, `Bearer`, and `API key` authentication are supported.
- Schemas defined using `anyOf`, `oneOf`, and `allOf` are currently not supported.

In addition, any call to the external API must complete within 30 seconds.

## Building the custom extension

To build a custom extension based on the API definition, follow these steps:

1. Go to the [Integrations](#) page.
2. Scroll to the **Extensions** section and click **Build custom extension**.
3. Read the **Get started** information and click **Next** to continue.
4. In the **Basic information** step, specify the following information about the extension you are creating:
  - **Extension name:** A short, descriptive name for the extension (for example, `CRM system` or `Weather service`). This is the name that will be displayed on the tile for the extension on the **Integrations** page, and in the list of available extensions in the action editor.
  - **Extension description:** A brief summary of the extension and what it does. The description will also be available from the **Integrations** page.

Click **Next**.

5. In the **Import OpenAPI** step, click or drag and drop to add the OpenAPI document that describes the REST API you want to integrate with.

If you encounter an error when you try to import the JSON file, make sure the file satisfies all of the requirements listed in [Preparing the API definition](#). Edit the file to correct errors or remove unsupported features. Click the **X** to clear the error message, and try the import again.

After you have imported the file successfully, click **Next**.

6. In the **Review extension** step, review what has been imported.

- The **Review authentication** table shows information about the authentication methods defined in the OpenAPI document. (Authentication methods are defined by the `securitySchemes` object in the OpenAPI document.)
- The **Review servers** table shows the URLs of the servers defined in the OpenAPI document, along with any server variables that must be specified.
- The **Review operations** table shows the operations that the assistant will be able to call from an action step. An *operation* is a request using a particular HTTP method, such as `GET` or `POST`, on a particular resource.

#### Review operations

This table shows the operations defined in the OpenAPI document.

Operation	Method	Resource
▼ List all pets	<code>GET</code>	<code>/pets</code>
▼ Create a pet	<code>POST</code>	<code>/pets</code>
▼ Info for a specific pet	<code>GET</code>	<code>/pets/{petId}</code>

For each operation, a row in the table shows the following information:

- **Operation:** A description of the operation, which is derived from either the `summary` (if present) or `description` in the OpenAPI file.
- **Method:** The HTTP method used to send the API request for the operation.
- **Resource:** The path to the resource the operation acts upon.

To see more information about an operation, click the `▼` icon next to its row in the table. The following additional details are shown:

- **Request parameters:** The list of input parameters defined for the operation, along with the type of each parameter and whether the parameter is required or optional.
- **Response properties:** The properties of the response body that will be mapped to variables the assistant can access.

7. If you are satisfied with the extension, click **Finish**.

If you want to change something, delete the extension, edit the JSON file to make your changes, and repeat the import process.

The new extension is now available as a tile in the **Extensions** section of the integrations catalog, and you can [add it to your assistant](#).

## Adding an extension to your assistant

After you build a custom extension, you must add it to the assistant before it can be accessed by actions.

Adding the extension to the assistant configures the extension for use within a particular environment, and it makes the extension available so that it can be called from actions.

You can use different configuration details for each environment. For example, you might want to use the URL for a test server in the draft environment, but a production server in the live environment.

For information about how to create a custom extension, see [Build a custom extension](#).

## Adding the extension to the draft environment

To add a custom extension to the assistant, follow these steps:

1. On the [Integrations](#) page, scroll to the **Extensions** section and find the tile for the custom extension you want to add.
2. Click **Add**. Review the overview of the extension and click **Confirm** to configure it for your assistant.

**Important:** When you first add an extension to an assistant, the configuration settings you provide are applied only to the draft environment. You must complete configuration for the draft environment before you can add the extension in the live environment.

3. Read the information in the **Get started** step, and then click **Next**.
4. In the **Authentication** step, specify the authentication and server information you want your assistant to use when calling the service.
  - In the **Authentication type** field, select the type of authentication to use.

Depending on the authentication type you select, you might also need to specify additional information (for example, **Username** and **Password** for HTTP basic authentication).

- In the **Servers** field, select the server URL to use.

If the selected URL contains any variables, also specify the values to use. Depending on how each variable is defined in the OpenAPI document, you can either select from a list of valid values or type the value to use in the field.

The **Generated URL** message shows the full URL that the assistant will use, including the variable values.

Click **Next**.

5. In the **Review extension** step, review the operations supported by the extension.

The **Review operations** table shows the operations that the assistant will be able to call from an action step. An *operation* is a request using a particular HTTP method, such as `GET` or `POST`, on a particular resource.

#### Review operations

This table shows the operations defined in the OpenAPI document.

Operation	Method	Resource
>List all pets	<code>GET</code>	/pets
Create a pet	<code>POST</code>	/pets
Info for a specific pet	<code>GET</code>	/pets/{petId}

For each operation, a row in the table shows the following information:

- **Operation:** A description of the operation, which is derived from either the `summary` (if present) or `description` in the OpenAPI file.
- **Method:** The HTTP method used to send the API request for the operation.
- **Resource:** The path to the resource the operation acts upon.

To see more information about an operation, click the  icon next to its row in the table. The following additional details are shown:

- **Request parameters:** The list of input parameters defined for the operation, along with the type of each parameter and whether the parameter is required or optional.
- **Response properties:** The properties of the response body that will be mapped to variables the assistant can

access.

6. Click **Finish**.

7. Click **Close** to return to the Integrations page.

The extension is now connected to your assistant and available for use by actions in the draft environment.

## Configuring the extension for the live environment

To configure the extension for the live environment, follow these steps:

1. On the  **Integrations** page, scroll to the **Extensions** section and find the tile for the custom extension you want to add.
2. Click **Open**. The **Open custom extension** window opens.
3. In the **Environment** field, select **Live**. Click **Confirm**.
4. Repeat the configuration process, specifying the values you want to use for the live environment.

 **Note:** If you are using multiple environments, follow the same steps to configure the extension for each environment.

For more information, see [Adding and using multiple environments](#).

The extension is now available in the environments you have configured, and it can be called from the assistant. For more information about how to call an extension from an action, see [Calling a custom extension](#).

## Plan limits

The number of custom extensions you can add to an assistant depends on your plan. If you already added the maximum number of extensions to the assistant, you must remove one before you can add a new one.

Plan	Extensions per assistant
Enterprise	100
Premium (legacy)	100
Plus	10
Trial	5
Lite	3
Standard (legacy)	0

Extension limits by plan

## Building a custom client

### Building a custom client using the API

If none of the built-in integrations meet your requirements, you can deploy your assistant by developing a custom client application that interacts with your users and communicates with the IBM Watson® Assistant service.

The Watson SDKs help you write code that interacts with Watson Assistant. For more information about the SDKs, see [Watson SDKs](#).

## Setting up the assistant

The example application we will create in this section implements several simple functions to illustrate how a client application interacts with Watson Assistant. The application code will collect input and send it to an assistant, which sends responses the application will display to the user.

If you want to try this example yourself, first you need to set up the simple example assistant that the client will connect to:

1. Download the actions [JSON file](#).
2. [Create an assistant](#).
3. In the new assistant, [open the global action settings](#). Go to the **Upload/Download** tab and import the actions from the file you downloaded.

The example actions include a *Greet customer* action that asks the customer's name, and simple actions for making and canceling appointments.

## Getting service information

To access the Watson Assistant REST APIs, your application needs to be able to authenticate with IBM Cloud® and connect to the assistant in the environment where it is deployed. You'll need to copy the service credentials and environment ID and paste them into your application code. You'll also need the URL for the location of your service instance (for example, <https://api.us-south.assistant.watson.cloud.ibm.com>).

To find this information, follow these steps:

1. Go to the **Environments** page and click the tab for the environment you want to connect to.
2. Click the  icon to open the environment settings.
3. Select **API details** to see details for the environment, including the service instance URL and environment ID. To find the API key, follow the link in the **Service credentials** section.

## Communicating with the Watson Assistant service

Interacting with the Watson Assistant service from your client application is simple. We'll start with an example that connects to the service, sends a single empty message, and prints the output to the console:

### Note

```
// Example 1: Creates service object, sends initial message, and
// receives response.

const AssistantV2 = require('ibm-watson/assistant/v2');
const { IamAuthenticator } = require('ibm-watson/auth');

// Create Assistant service object.
const assistant = new AssistantV2({
  version: '2021-11-27',
  authenticator: new IamAuthenticator({
    apikey: '{apikey}', // replace with API key
  }),
  url: '{url}', // replace with URL
});

const assistantId = '{environment_id}'; // replace with environment ID
```

```

// Start conversation with empty message
messageInput = {
  messageType: 'text',
  text: '',
};
sendMessage(messageInput);

// Send message to assistant.
function sendMessage(messageInput) {
  assistant
    .messageStateless({
      assistantId,
      input: messageInput,
    })
    .then(res => {
      processResult(res.result);
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
}

// Process the result.
function processResult(result) {
  // Print responses from actions, if any. Supports only text responses.
  if (result.output.generic) {
    if (result.output.generic.length > 0) {
      result.output.generic.forEach( response => {
        if (response.response_type == 'text') {
          console.log(response.text);
        }
      });
    }
  }
}

```

## Python

```

# Example 1: Creates service object, sends initial message, and
# receives response.

from ibm_watson import AssistantV2
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

# Create Assistant service object.
authenticator = IAMAuthenticator('{apikey}') # replace with API key
assistant = AssistantV2(
    version = '2021-11-27',
    authenticator = authenticator
)
assistant.set_service_url('{url}') # replace with service instance URL
assistant_id = '{environment_id}' # replace with environment ID

# Start conversation with empty message.
result = assistant.message_stateless(
    assistant_id,
).get_result()

# Print responses from actions, if any. Supports only text responses.
if result['output']['generic']:
    for response in result['output']['generic']:
        if response['response_type'] == 'text':
            print(response['text'])

```

The first step is to create a service object, a sort of wrapper for the Watson Assistant service.

You use the service object for sending input to, and receiving output from, the service. When you create the service object, you

specify the API key for authentication, as well as the version of the Watson Assistant API you are using.

In this Node.js example, the service object is an instance of `AssistantV2`, stored in the variable `assistant`. The Watson SDKs for other languages provide equivalent mechanisms for instantiating a service object.

In this Python example, the service object is an instance of `watson_developer_cloud.AssistantV2`, stored in the variable `assistant`. The Watson SDKs for other languages provide equivalent mechanisms for instantiating a service object.

After creating the service object, we use it to send a message to the assistant, using the stateless `message` method. In this example, the message is empty; we just want to trigger the *Greet customer* action to start the conversation, so we don't need any input text. We then print any text responses returned in the `generic` array in the returned output.

Use the `node <filename.js>` command to run the example application.

Use the `python3 <filename.py>` command to run the example application.

**Note:** Make sure you have installed the Watson SDK for Node.js using `npm install ibm-watson`.

**Note:** Make sure you have installed the Watson SDK for Python using `pip install --upgrade ibm-watson` or `easy_install --upgrade ibm-watson`.

Assuming everything works as expected, the assistant returns the output from the assistant, which the app then prints to the console:

```
Welcome to the Watson Assistant example. What's your name?
```

This output tells us that we have successfully communicated with the assistant and received the greeting message specified by the *Greet customer* action. But we don't yet have a way of responding to the assistant's question.

## Processing user input

To be able to process user input, we need to add a user interface to our client application. For this example, we'll keep things simple and use standard input and output. We can use the Node.js `prompt-sync` module to do this. (You can install `prompt-sync` using `npm install prompt-sync`.) We can use the Python 3 `input` function to do this.

### Node

```
// Example 2: Adds user input.

const prompt = require('prompt-sync')();
const AssistantV2 = require('ibm-watson/assistant/v2');
const { IamAuthenticator } = require('ibm-watson/auth');

// Create Assistant service object.
const assistant = new AssistantV2({
  version: '2021-11-27',
  authenticator: new IamAuthenticator({
    apikey: '{apikey}', // replace with API key
  }),
  url: '{url}', // replace with URL
});

const assistantId = '{environment_id}'; // replace with environment ID

// Start conversation with empty message
messageInput = {
  messageType: 'text',
  text: '',
};

sendMessage(messageInput);
```

```

// Send message to assistant.
function sendMessage(messageInput) {
  assistant
    .messageStateless({
      assistantId,
      input: messageInput,
    })
    .then(res => {
      processResult(res.result);
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
}

// Process the result.
function processResult(result) {

  // Print responses from actions, if any. Supports only text responses.
  if (result.output.generic) {
    if (result.output.generic.length > 0) {
      result.output.generic.forEach( response => {
        if (response.response_type === 'text') {
          console.log(response.text);
        }
      });
    }
  }

  // Prompt for the next round of input unless skip_user_input is true.
  let newMessageFromUser = '';
  if (result.context.global.system.skip_user_input !== true) {
    newMessageFromUser = prompt('>> ');
  }

  if (newMessageFromUser !== 'quit') {
    newMessageInput = {
      messageType: 'text',
      text: newMessageFromUser,
    }
    sendMessage(newMessageInput);
  }
}

```

## Python

```

# Example 2: Adds user input.

from ibm_watson import AssistantV2
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

# Create Assistant service object.
authenticator = IAMAuthenticator('{apikey}') # replace with API key
assistant = AssistantV2(
  version = '2021-11-27',
  authenticator = authenticator
)
assistant.set_service_url('{url}') # replace with service instance URL
assistant_id = '{environment_id}' # replace with environment ID

# Initialize with empty value to start the conversation.
message_input = {
  'message_type': 'text',
  'text': ''
}

# Main input/output loop

```

```

while message_input['text'] != 'quit':

    # Send message to assistant.
    result = assistant.message_stateless(
        assistant_id,
        input = message_input
    ).get_result()

    # Print responses from actions, if any. Supports only text responses.
    if result['output']['generic']:
        for response in result['output']['generic']:
            if response['response_type'] == 'text':
                print(response['text'])

    # Prompt for the next round of input unless skip_user_input is True.
    if not result['context']['global']['system'].get('skip_user_input', False):
        user_input = input('>> ')
        message_input = {
            'text': user_input
        }

```

This version of the application begins the same way as before: sending an empty message to the assistant to start the conversation.

The `processResult()` function displays the text of any responses received from the assistant. It then prompts for the next round of user input.

It then displays the text of any responses received from the assistant, and it prompts for the next round of user input.

**Tip:** The example checks for the global context variable `skip_user_input` and prompts for user input only if this variable is not set to true `True`. The `skip_user_input` variable is set by the assistant in some situations where no user input is needed (for example, if the assistant has called an external service but is still waiting for the result). It's good practice always to make this check before prompting for user input.

Because we need a way to end the conversation, the client app is also watching for the literal command `quit` to indicate that the program should exit.

But something still isn't right:

```

Welcome to the Watson Assistant example. What's your name?
>> Robert
I'm afraid I don't understand. Please rephrase your question.
>> I want to make an appointment.
What day would you like to come in?
>> Thursday
I'm afraid I don't understand. Please rephrase your question.
>>

```

The assistant is starting out with the correct greeting, but it doesn't understand when you tell it your name. And if you tell it you want to make an appointment, the correct action is triggered; but once again, it doesn't understand when you answer the follow-up question.

This is happening because we are using the stateless `message` method, which means that it is the responsibility of our client application to maintain state information for the conversation. Because we are not yet doing anything to maintain state, the assistant sees every round of user input as the first turn of a new conversation. Because it has no memory of asking a question, it tries to interpret your answer as a new question or request.

## Maintaining state

State information for your conversation is maintained using the `context`. The context is an object that is passed back and forth between your application and the assistant, storing information that can be preserved and updated as the conversation goes on. Because we are using the stateless `message` method, the assistant does not store the context, so it is the responsibility of our client application to maintain it from one turn of the conversation to the next.

The context includes a session ID for each conversation, as well as a counter that is incremented with each turn of the conversation. The assistant updates the context and returns it with each response. But our previous version of the example did not preserve the context, so these updates were lost, and each round of input appeared to be the start of a new conversation. We can fix that by saving the context and sending it back to the assistant each time.

In addition to maintaining our place in the conversation, the context can also contain action variables that store any other data you want to pass back and forth between your application and the assistant. This can include persistent data you want to maintain throughout the conversation (such as a customer's name or account number), or any other data you want to track (such as the contents of a shopping cart or user preferences).

## Node

```
// Example 3: Preserves context to maintain state.

const prompt = require('prompt-sync')();
const AssistantV2 = require('ibm-watson/assistant/v2');
const { IamAuthenticator } = require('ibm-watson/auth');

// Create Assistant service object.
const assistant = new AssistantV2({
  version: '2021-11-27',
  authenticator: new IamAuthenticator({
    apikey: '{apikey}', // replace with API key
  }),
  url: '{url}', // replace with URL
});

const assistantId = '{environment_id}'; // replace with environment ID

// Start conversation with empty message
messageInput = {
  messageType: 'text',
  text: '',
};
context = {};
sendMessage(messageInput);

// Send message to assistant.
function sendMessage(messageInput, context) {
  assistant
    .messageStateless({
      assistantId,
      input: messageInput,
      context: context,
    })
    .then(res => {
      processResult(res.result);
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
}

// Process the result.
function processResult(result) {
  let context = result.context;
```

```

// Print responses from actions, if any. Supports only text responses.
if (result.output.generic) {
  if (result.output.generic.length > 0) {
    result.output.generic.forEach( response => {
      if (response.response_type === 'text') {
        console.log(response.text);
      }
    });
  }
}

// Prompt for the next round of input unless skip_user_input is true.
let newMessageFromUser = '';
if (result.context.global.system.skip_user_input !== true) {
  newMessageFromUser = prompt('>> ');
}

if (newMessageFromUser !== 'quit') {
  newMessageInput = {
    messageType: 'text',
    text: newMessageFromUser,
  }
  sendMessage(newMessageInput, context);
}
}

```

## Python

```

# Example 3: Preserves context to maintain state.

from ibm_watson import AssistantV2
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

# Create Assistant service object.
authenticator = IAMAuthenticator('{apikey}') # replace with API key
assistant = AssistantV2(
    version = '2021-11-27',
    authenticator = authenticator
)
assistant.set_service_url('{url}') # replace with service instance URL
assistant_id = '{environment_id}' # replace with environment ID

# Initialize with empty message to start the conversation.
message_input = {
    'message_type': 'text',
    'text': ''
}
context = {}

# Initialize with empty message to start the conversation.
message_input = {
    'message_type': 'text',
    'text': ''
}
context = {}

# Main input/output loop
while message_input['text'] != 'quit':

    # Send message to assistant.
    result = assistant.message_stateless(
        assistant_id,
        input = message_input,
        context = context
    ).get_result()

    context = result['context']

```

```

# Print responses from actions, if any. Supports only text responses.
if result['output']['generic']:
    for response in result['output']['generic']:
        if response['response_type'] == 'text':
            print(response['text'])

# Prompt for the next round of input unless skip_user_input is True.
if not result['context']['global']['system'].get('skip_user_input', False):
    user_input = input('>> ')
    message_input = {
        'text': user_input
    }

```

The only change from the previous example is that we are now storing the context received from the assistant in a variable called `context`, and we're sending it back with the next round of user input:

The only change from the previous example is that we are now storing the context received from the assistant in a variable called `context`, and we're sending it back with the next round of user input:

## Node

```

assistant
  .messageStateless({
    assistantId,
    input: messageInput,
    context: context,
  })

```

## Python

```

response = assistant.message_stateless(
    assistant_id,
    input = message_input,
    context = context
).get_result()

```

This ensures that the context is maintained from one turn to the next, so the Watson Assistant service no longer thinks every turn is the first:

```

Welcome to the Watson Assistant example. What's your name?
>> Robert
Hi, Robert! How can I help you?
>> I want to make an appointment.
What day would you like to come in?
>> Next Monday
What time works for you?
>> 10 AM
OK, Robert. You have an appointment for 10:00 AM on Sep 12. See you then!

```

Success! The application now uses the Watson Assistant service to understand natural-language input, and it displays the appropriate responses.

This simple example illustrates how you can build a custom client app to communicate with the assistant. Of course, a real-world application would use a more sophisticated user interface, and it might integrate with other applications such as a customer database or other business systems. It would also need to send additional data to the assistant, such as a user ID to identify each unique user. But the basic principles of how the application interacts with the Watson Assistant service would remain the same.

## Using the v1 runtime API

Using the v2 API is the recommended way to build a runtime client application that communicates with the Watson Assistant service. However, some older applications might still be using the v1 runtime API, which includes a similar method for sending messages to the workspace within a dialog skill. Note that if your app uses the v1 runtime API, it communicates directly with the workspace, bypassing the skill orchestration and state-management capabilities of the assistant.

For more information about the v1 `/message` method and context, see the [v1 API Reference](#).

## Watson SDKs

SDKs abstract much of the complexity associated with application development. By providing programming interfaces in languages that you already know, they can help you get up and running quickly with IBM Watson services.

## Supported SDKs

The following Watson SDKs are supported by IBM:

- [Java SDK](#)
- [Node.js SDK](#)
- [Python SDK](#)
- [.NET SDK](#)

**Tip:** The [API reference](#) for each service includes information and examples for these SDKs.

## Community SDKs

The following SDKs are available from the Watson community of developers:

- [ABAP SDK for IBM Watson](#), using SAP NetWeaver
- [Android SDK](#)
- [Go SDK](#)
- [Ruby SDK](#)
- [Salesforce SDK](#)
- [Swift SDK](#)
- [Unity SDK](#)

## SDK updates and deprecation

The supported Watson SDKs are updated according to the following guidelines.

### Semantic versioning

Supported Watson SDKs adhere to semantic versioning with releases labeled as `{major}.{minor}.{patch}`.

### Release frequency

SDKs are released independently and might not update on the same schedule.

- The current releases of the Watson SDKs are updated on a 2- to 6-week schedule. These releases are either minor updates or patches that do not include breaking changes. You can update to any version of the SDK with the same major version number.
- Major updates that might include breaking changes are released approximately every 6 months.

### Deprecated release

When a major version is released, support continues on the previous major release for 12 months in a deprecation period. The deprecated release might be updated with bug fixes, but no new features will be added and documentation might not be

available.

### **Obsolete release**

After the 12-month deprecation period, a release is obsolete. The release might be functional but is unsupported and not updated. Update to the current release.

# Migrating to the new experience

## Administering your instance

---

These topics cover several tasks and areas involved with administering your instance of IBM Watson® Assistant.

Topic	Description
<a href="#">Managing access</a>	You can give other people access to your Watson Assistant instance and resources, and control the level of access they get.
<a href="#">Managing your plan</a>	A Watson Assistant plan information reference and steps on upgrading your plan.
<a href="#">Auditing user activity</a>	As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with Watson Assistant.
<a href="#">Securing your assistant</a>	Data privacy, security, and governance solutions.
<a href="#">Backing up and restoring data</a>	Back up and restore your data by downloading, and then uploading the data.
<a href="#">High availability and disaster recovery</a>	Watson Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
<a href="#">Failover options</a>	This topic explains some actions you can take to increase the availability of Watson Assistant for your organization.
<a href="#">Early access program</a>	When you participate in the early access program, IBM gives you early access to features for your evaluation.
<a href="#">Supported languages</a>	Watson Assistant supports individual features to varying degrees per language.

## Comparing actions and dialog

---

Choose the right type of conversation for your use case.

### Actions benefits

Using actions is the best choice when you want to approach the assistant with a focus on content. Actions offers the following benefits:

- The process of creating a conversational flow is easier. People who have expertise with customer care can write the words that your assistant says. With a simplified process anyone can build a conversation. You don't need knowledge about machine learning or programming.
- Actions provide better visibility into the customer's interaction and satisfaction with the assistant. Because each task is discrete and has a clear beginning and ending, you can track user progress through a task and identify snags.
- The conversation designer doesn't need to manage data collected during the conversation. By default, your assistant collects and stores information for the duration of the current action. You don't need to take extra steps to delete saved data or reset the conversation. But if you want, you can store certain types of information, such as the customer's name, for the duration of a conversation.
- Many people can work at the same time in separate, self-contained actions. The order of actions within a conversation doesn't matter. Only the order of steps within an action matters. And the action author can use drag and drop to

reorganize steps in the action for optimal flow.

## Dialog benefits

A dialog-based conversation is the best choice when you want greater control over the logic of the flow. The dialog editor exposes more of the underlying artifacts (such as intents and entities) used to build the AI models. The dialog flow uses an if-then-else style structure that might be familiar to developers, but not to content designers or customer-care experts.

## How actions are different from dialog

If you are already familiar with dialog-based conversations, learn more about how actions compares.

Feature	Actions	Dialog
<b>Automatic reset of context</b>	<input type="checkbox"/>	
<b>Keep track of context</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Collect info, as with slots</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Contextual entities</b>		<input type="checkbox"/>
<b>Collect numbers (@sys-number detection)</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Detection of other system entities</b>		<input type="checkbox"/>
<b>Connect to agent response type</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Free text response type</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Image response type</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Options response type</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Search skill response type</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Rich text editor for text responses</b>	<input type="checkbox"/>	
<b>User input validation</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Step logic validation</b>	<input type="checkbox"/>	
<b>Support multiple users by notifying them when simultaneous edits are made to the skill</b>	<input type="checkbox"/>	
<b>Use SpEL expressions</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Disambiguation</b>	<input type="checkbox"/>	<input type="checkbox"/>

<b>Digression support</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Spelling correction</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Webhook (before or after every message) support</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Webhook (from a node) support</b>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Webhook (log all messages) support</b>	<input type="checkbox"/>	<input type="checkbox"/>

#### Conversational flow skill feature support

For some functions, there is parity but you follow different steps to implement the behavior you want.

- **Jump-to:** In actions, you can jump from one step to another. In a dialog, you use a jump-to to skip to a specific dialog node in the same branch of the conversation. With actions, you can jump to a different step within an action also. However, to do so, you use conditions on the intervening steps to prevent them from being processed, rather than using an explicit jump-to. The benefit of this approach is that it's easier to anticipate the path of a conversation and follow it later if there are not multiple jump-tos sprinkled throughout the flow.
- **Slots:** In a dialog, you add slots to a dialog node to call out a set of values that you want to collect from the user, and that you will take and store in any order. In actions, every step in the action acts like a slot. If the user provides information that address step 10 when answering the question to step 1, both step 1 and step 10 are filled. In fact, if you want step 10 to ask the question explicitly, you must select the **Always ask for this** option on step 10.

**Tip:** *Want to get started with actions, but need features that are available from a dialog?* Use both. Dialog is your primary conversation with users, but you can call an action from your dialog to perform a discrete task. For more information, see [Calling an actions from a dialog](#).

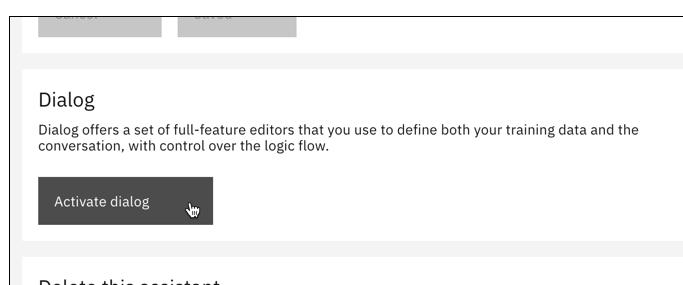
## Activating dialog and migrating skills

To use an existing dialog skill, you need to activate the dialog feature in your assistant. Then you can upload your existing skill.

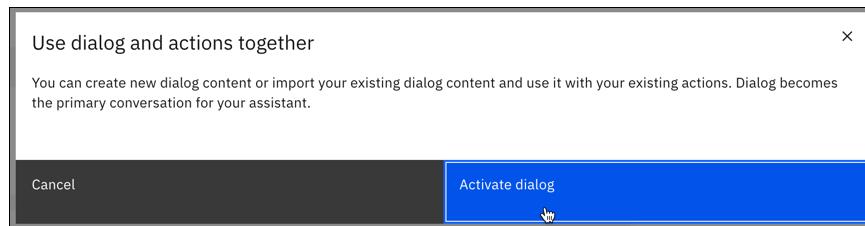
### Activating dialog

To activate the dialog feature in the new experience:

1. Create or open an assistant where you want to use a dialog as the primary conversation with users.
2. Open **Assistant settings** .
3. Click **Activate dialog**.



4. In the confirmation that displays, click **Activate dialog** again.



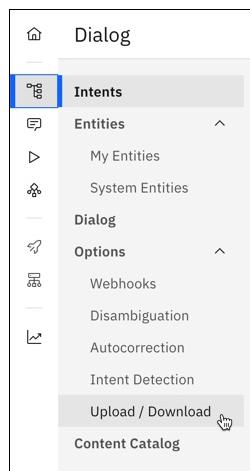
Once you activate the dialog feature, it takes precedence over actions. You can use actions to supplement a dialog-based conversation, but the dialog drives the conversation with users to match their requests. For more information, see [Calling actions from a dialog](#).

## Migrating existing skills

Once the dialog feature is activated, you can start a new dialog conversation from scratch if you want. But, you probably have an existing dialog skill that you want to migrate into the new Watson Assistant.

To migrate an existing dialog skill:

1. Use the classic experience to [download your dialog skill](#) in JSON format.
2. In the new Watson Assistant, open the **Dialog** page.
3. In **Options**, choose **Upload / Download**.



4. On the **Upload** tab, upload the JSON file for your dialog skill.

## Migrating intents and entities

You can migrate your intents and entities from the classic Watson Assistant experience to the new Watson Assistant experience.

### Downloading intents

You can download intents to a CSV file, so you can then upload and reuse them as actions or example phrases in the new Watson Assistant experience.

1. Go to the **Intents** page.
2. Select the intents that you want to download, and then click the **Download** icon .

**⚠Important:** If you plan to use a downloaded intent file to upload example phrases for a specific action, download only a single intent. In the new experience, only one intent can be uploaded per action.

3. Specify the name and location in which to store the CSV file that is generated, and click **Save**.

You can now perform the following tasks to migrate information to the new Watson Assistant:

- Upload intents as actions. For more information, see [Uploading intents as actions](#).
- Upload the examples from a single intent as example phrases for a specific action. For more information, see [Uploading phrases](#).

## Downloading entities

You can download a number of entities to a CSV file, so you can then upload and reuse them as saved customer responses in the new Watson Assistant experience.

1. Go to the **Entities** page.
2. Select the entities that you want to download, and then click the **Download** icon .
3. Specify the name and location in which to store the CSV file that is generated, and click **Save**.

You can now upload the entities as saved customer responses in the new Watson Assistant. For more information, see [Uploading saved customer responses](#).

## Calling actions from a dialog

In the new Watson Assistant, you can use actions with your primary dialog conversation. A dialog feature takes precedence over actions. You can use actions to supplement a dialog-based conversation, but the dialog drives the conversation with users to match their requests.

A dialog node calls an action to perform a task and then return to the dialog. From a single dialog node, you can make a call to either a webhook or an action, not both. Choose the approach that best suits your needs:

- Call an action if you want to perform a discrete and self-contained task, and then return to the dialog to be ready to address any new user requests. Data collected from the user during action processing must be stored as a session variable for it to be passed to the dialog.
- Call a webhook if you want to send a request to a web service to complete a task and return a response that can be used later by this dialog. For more information, see [Extending your assistant with webhooks](#).

## When to call an action from a dialog

You might want to call an action from a dialog in the following scenarios:

- You want to perform an action from multiple dialog threads.

For example, you want to ask customers how satisfied they are with your service. You can define a single action to check customer satisfaction and call it from multiple branch-ending dialog nodes.

In this scenario, you don't need to define an intent, such as `#check_satisfaction`. The action is called automatically, replacing a jump to another dialog node's response.

- You want to see how actions works.

In this scenario, you can pick an intent for the action to handle. If you plan for the action to be called from the dialog only, you can spend your time defining the user examples for the intent that triggers the dialog node. When you define the action, you can add one customer example only, reducing the time you need to spend building the action.

## Adding a call to an action

Before you start, open the **Actions** page and check the following:

- The name of the action you want to call.
- If the action you want to call uses a session variable, and you want to pass a value to the action by setting the session variable value, make a note of the session variable's ID.

In the Variables section, click to open the session variable. You can see the syntax that is used for the variable name in the **ID** field. Copy the variable ID to the clipboard.

You also need to understand the type of value to assign to the session variable. If you don't know, check how the session variable is used by the action.

To call an action from a dialog node:

1. From the dialog node, click **Customize**.
2. Set **Call out to webhooks / actions** to **On**.
3. Select **Call an action**.

When you add a call to an action, multiple conditional responses are enabled for the dialog node automatically.

4. Click **Apply**.
5. In the dialog node, choose the action you want to call.

6. **Optional:** If the action you want to call uses a session variable, you can set the value of the session variable when the action is started.

In the **Parameters** section, add a key and value pair to pass with the call to the action.

Add the variable ID (that you copied to the clipboard earlier) to the **Key** field, and in the **Value** field, specify the value that you want to set the variable to.

For example, let's say the action has a session variable named `given_name`. You can pass the current user's given name (which you asked for and saved to a `$name` context variable) to the action as follows:

Key	Value
<code>given_name</code>	<code>"\$name"</code>

**Action call key and value pair example**

7. If you call actions from more than one dialog node, change the name of the return variable to make it unique across all of your dialog nodes. For example, you might already call an action and its return variable is called `$action_result_1`, so you can name the new one `$action_result_2`.

The return variable is a context variable that is automatically created. It stores any session variable values that are created or have their values changed while the called action is processed.

8. In the **Assistant responds** section, the condition for the first conditional response is populated automatically with the return variable from the action call.

The return variable is named `$action_result_1` unless you change the name.

If any session variables are created or have their values changed by the called action, then they are returned and stored in the result variable as a JSON object.

For example, if the action changes the value of the `given_name` and `membership_status` session variables, the following JSON object is returned:

```
{"given_name": "Sally", "membership_status": "true"}
```

9. If you don't send or change any session variables, you can delete the first conditional response. Otherwise, consider adding a response to show when a return variable is sent from the called action.

You might want to specify a custom response to show in case the exchange that took place when the action was processed resulted in a change to the value of the session variable that you passed to the action.

For example, let's say that the dialog asks for the person's given name and stores it in the `$name` context variable. The dialog then passes the name to the action as a `given_name` session variable. Then, the action that is called asks if the customer prefers that the assistant use a nickname. The action then replaces the value that was stored in the `given_name` session variable with the nickname that the customer submitted.

To continue with this example, you might want to address the user by the preferred nickname, now that you know it. You can add a response that says, `Thanks for your business, <? $action_result_1.given_name ?>!` The `<? $action_result_1.given_name ?>` expression extracts the value of the `given_name` session variable that is returned from the called action.

10. Add a response to show when no return variable is provided as the second conditional response.

The second response that is added automatically for you has an `anything_else` condition. This response is shown if none of the other conditional responses are displayed.

You can delete or edit the conditional responses that are added automatically. You can add more conditional responses and reorder them.

11. Click **X** to close the dialog node. Your changes are saved automatically.

12. Use the **Preview** page to test the interaction between the dialog and the action.

## Analyzing dialog and actions

---

The **Analyze** page provides a summary of the interactions between users and your assistant. If the dialog feature is enabled, the **Analyze** page remains the same, but some slight differences in functionality exist.

### Overview tab

When you view the **Overview** page, you can see action completion information in the **Action completion** diagram if a dialog node triggers an action. The **Action completion** diagram is empty if you are using only a dialog in your assistant. The three cards that display information about the most frequent actions, least frequent actions, and least completed actions are not available if your assistant uses only a dialog.

For more information about the **Analyze** page and how to use analytics with actions, see [Use analytics to review your entire](#)

[assistant at a glance](#).

## Action completion tab

The **Action completion** page of Watson Assistant provides an overview of how all your assistant's actions are doing. If the dialog feature is enabled, the **Action completion** tab is relevant only if a dialog node triggers an action. If your assistant uses only a dialog, then this tab will be empty.

For more information about understanding action completion with actions, see [Understand your most and least successful actions](#).

## Conversations tab

The **Conversations** page of Watson Assistant provides a history of conversations between users and a deployed assistant. You can use this history to improve how your assistants understand and respond to user requests.

From the **Conversations** page, an intent that directly calls an action is displayed in the **Topics** column. For example, you might set up an intent called `#buy_takeout` in the dialog, and that intent calls the `order pizza` action. This conversation topic is listed as `#buy_takeout > order pizza` in the **Topics** column.

You might also see **Dialog called action** listed in the **Requests** column next to a conversation. In this case, customer input triggered an intent. Then, the customer engaged with the assistant before an action was eventually called.

For more information about analyzing conversations with actions, see [Review customer conversations](#).

## Publishing dialog and actions

---

If the dialog feature is enabled, the publishing and deployment processes remain the same. However, some slight differences in functionality exist.

To learn about the overall publishing and deployment model for Watson Assistant, see the [Publishing and deploying your assistant overview](#). For more information about the **Publish** page and how the publishing process works, see [Publishing your content](#). The following slight differences exist when the dialog feature is enabled in an assistant:

- On the **Publish** page, the information in the **Content type** column lists whether your content changes contain a dialog.
- The version tiles on the **Publish** and **Environments** pages show whether the published content contains actions, or actions and a dialog. For example, if the dialog feature is enabled in your assistant, the version tile displays `Contains actions & dialog`.
- When you export a version of your content from the **Publish** page, two JSON files are downloaded. One file is for actions and one file is for the dialog.

# Early access features

**Note:** This feature is available only to users enrolled in the early access program. To gain access to early access features, complete the [Early access to beta features form](#).

## Action response modes

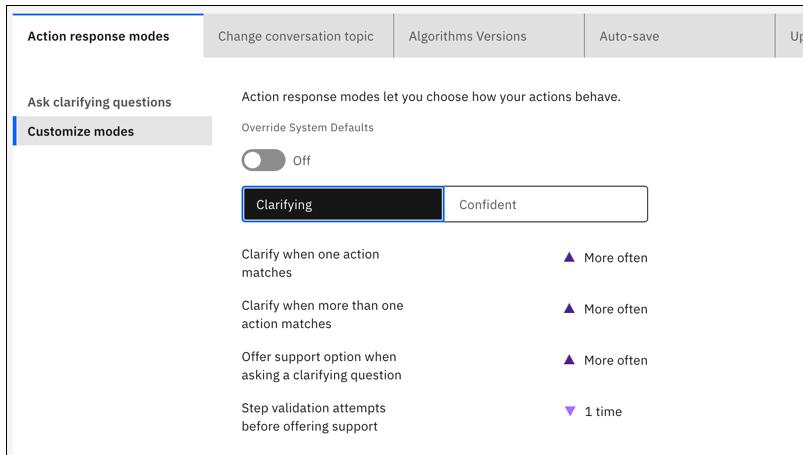
You can choose a response mode for each action. This lets you set how each action behaves. There are two modes:*clarifying* and *confident*.

**Clarifying mode:** Start here. In the clarifying mode, your assistant is eager to ask questions so you can ensure your customer gets to the action they need. An assistant is more likely to ask questions to be sure an action matches what a customer is asking. This lets you be sure a new or untested action gets the training it needs.

**Confident mode:** Take the next step. After you use analytics to improve your assistant, use the confident mode. Your assistant solves customer issues with authority and accuracy. An assistant is less likely to ask questions and is more likely to trigger actions that match. Use confident mode when you have tested and trained actions.

## Settings

Settings for the two modes are in the global settings. For more information, see [Global settings for actions](#).



The settings are:

**Clarify when one action matches:** If an assistant prioritizes one action that it thinks matches the customer need, it can clarify the match by asking the customer to confirm. This helps you ensure the action is the right one and gives the customer a chance to give input before proceeding. For example, if the assistant thinks that a single action named `Pay Bill` is the right one, it can clarify the choice by asking the customer `Did you mean: Pay Bill`.

**Clarify when more than one action matches:** When your assistant finds that more than one action might fulfill a customer's request, it can automatically ask for clarification. Instead of guessing which action to use, your assistant shows a list of the possible actions to the customer and asks the customer to pick the right one.

**Offer support option when asking a clarifying question:** When asking for clarification, the assistant can include a choice to connect to other support. If the customer picks this choice, the assistant uses your Fallback action.

**Step validation attempts before offering support:** If a customer provides invalid answers for a step in an action, the assistant can offer to connect to other support in the Fallback action. The step validation count measures how many invalid answers can

occur before providing this choice.

This table shows the default settings for each mode.

	<b>Clarifying</b>	<b>Confident</b>
Clarify when one action matches	More often	Sometimes
Clarify when more than one action matches	More often	Sometimes
Offer support option when asking a clarifying question	More often	Sometimes
Step validation attempts before offering support	1 time	3 times

Default settings

## Customize modes Enterprise

For Lite and Plus plans, the default settings can't be changed. For Enterprise plans, you can customize each mode in global settings for actions.

If you customize, here are the choices for each setting:

<b>Mode</b>	<b>Choices</b>
Clarify when one action matches	More often, Sometimes, Less often
Clarify when more than one action matches	More often, Sometimes, Less often
Offer support option when asking a clarifying question	More often, Sometimes, Less often
Step validation attempts before offering support	1 time, 2 times, 3 times

Mode customization choices



**Note:** If you customize modes, be sure to test any changes.

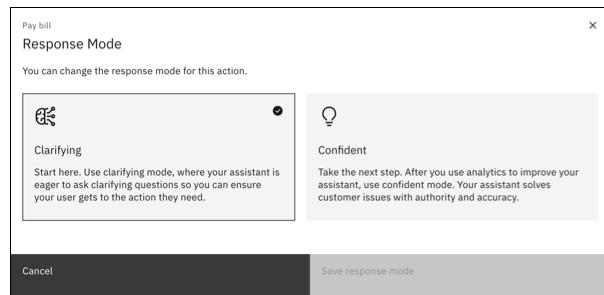
## Default for new actions

In global settings for actions, you can choose what mode to use when you create a new action. Clarifying mode is the default and is designed for use with new, untested actions that need training.

## Choosing a mode for individual actions

When editing an action, you can see the mode it uses and change it if you need to.

1. Click the Action response mode icon. The mode in use is checked.



2. Click the other mode if you want to change it, and then click **Save response mode**.

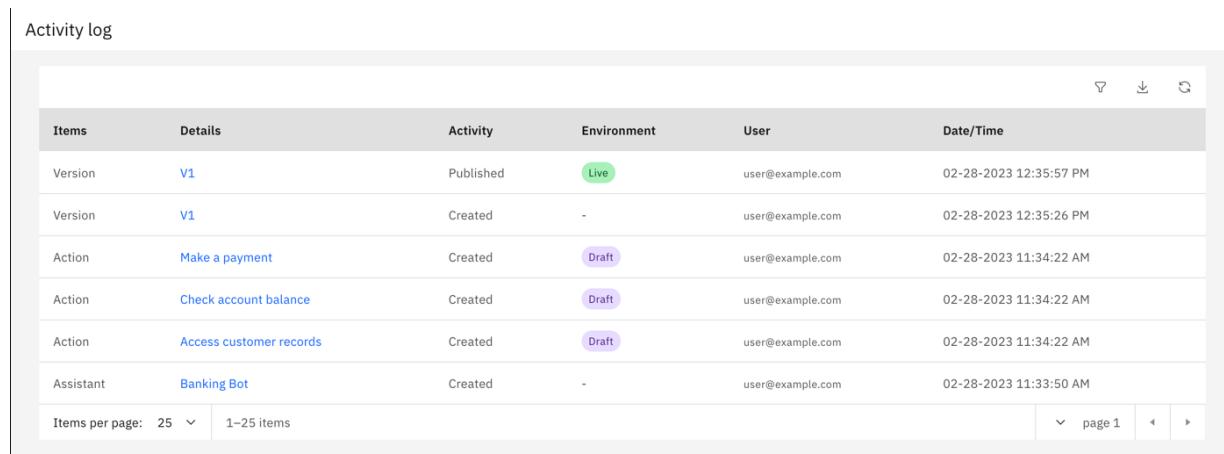


**Note:** This feature is available only to users enrolled in the early access program. To gain access to early access features, complete the [Early access to beta features form](#).

## Activity log

PlusEnterprise

Use the *activity log* to track changes made in your assistant. It gives you visibility into the modifications that are made to your assistant. It is available for Plus plans and higher.



Items	Details	Activity	Environment	User	Date/Time
Version	<a href="#">V1</a>	Published	Live	user@example.com	02-28-2023 12:35:57 PM
Version	<a href="#">V1</a>	Created	-	user@example.com	02-28-2023 12:35:26 PM
Action	<a href="#">Make a payment</a>	Created	Draft	user@example.com	02-28-2023 11:34:22 AM
Action	<a href="#">Check account balance</a>	Created	Draft	user@example.com	02-28-2023 11:34:22 AM
Action	<a href="#">Access customer records</a>	Created	Draft	user@example.com	02-28-2023 11:34:22 AM
Assistant	<a href="#">Banking Bot</a>	Created	-	user@example.com	02-28-2023 11:33:50 AM

## Activity log

Available days of activity depend on your plan.

Plan	Days of activity
Plus	30
Enterprise	90
Premium	90

## User access to the activity log

To access and view the log, users need at least the *Reader* service role for an assistant. Users might not be able to open links for specific events, depending on their access to all of the items in an assistant.

For more information, see [Managing access](#).

## Using the activity log

To open the log, click the **Activity log** icon .

The activity log includes these columns of data:

Column	Description
Item	Describes what was impacted by the change that was made within the assistant. For more information, see <a href="#">Items and activity tracked</a> .
Details	Shows details about the item that was changed. For example, for an action, this column lists the specific action name and provides a link to open it.
Activity	Describes the change that was made. For more information, see <a href="#">Items and activity tracked</a> .
Environment	The environment in which the activity took place or the end destination of an activity.
User	Email address of the user that made the change.
Date/Time	Date and time of the activity in the format DD-MM-YYYY hr:min:sec AM/PM

Activity log columns

## Filtering

Click the **Filter** icon  to filter the log. You can filter on:

- Item
- Details
- Activity
- Environment
- User (requires exact match)
- Date range

## Sorting

To sort the log, click either the **Details** or **Date/Time** columns.

## Downloading

You can download the activity log as a CSV file. You can either download all activity, or filter the activity log by any of [filtering](#) choices.

To download the activity log, click the **Download** icon .

## Items and activity tracked

For the early access release, here's the current list of items and activity that are tracked. Events are tracked if they complete successfully.

Item	Activity	Notes
Assistant	Created	
Assistant Settings	Updated	
Action	Created, Updated, Deleted	<ul style="list-style-type: none"> <li>Each save (auto or manual) is tracked as an update.</li> <li>Changes to action settings are tracked as an update.</li> </ul>
Variable	Created, Updated, Deleted	
Saved Response Type	Created, Updated, Deleted	
Version	Created, Published, Deleted	
Environment	Created, Updated, Deleted	Dialog activation is tracked as an environment update.
Intent	Created, Updated, Deleted	<ul style="list-style-type: none"> <li>Each save (auto or manual) is tracked as an update.</li> <li>A renamed intent retains the old name in previous entries and won't have a link.</li> <li>Only individual deletions are included. Bulk deletions are not.</li> </ul>
Entity	Created, Updated, Deleted	<ul style="list-style-type: none"> <li>Each save (auto or manual) is tracked as an update.</li> <li>A renamed entity retains the old name in previous entries and won't have a link.</li> <li>Only individual deletions are included. Bulk deletions are not.</li> </ul>
Dialog Node	Created, Updated, Deleted	Each save (auto or manual) is tracked as an update.
Search	Updated	

#### Items and activity

 **Note:** If an item was triggered by the API, the activity log shows the account owner of the API key, even if the key is shared with another person who is triggered the event.

## What isn't included in the log

These items aren't included in the activity log:

- Copying an action from another assistant
- Uploading actions, intents, entities aren't included as created
- Changing actions global settings
- Reverting a version to draft

- Activating or deactivating dialog (this is included as an update to environments)
- Adding or deleting entity values and synonyms
- Bulk deletion of intents or entities
- Creating, updating, or deleting channel integrations
- Creating, updating, or deleting custom extensions



**Note:** This feature is available only to users enrolled in the early access program. To gain access to early access features, complete the [Early access to beta features form](#).

## Autolearning

Use *autolearning* to enable your assistant to learn from interactions with your customers and improve responses.



**Note:** This beta feature is available in English-language assistants only.

When customers interact with your assistant, they often make choices. Your assistant can learn from these user decisions.

For example, a customer might ask a question that the assistant isn't sure it understands. The assistant asks a clarifying question so the customer can choose the right action from a list. If customers most often click the same action (option #2, for example), your assistant can learn that option #2 is the best answer.

Next time, the assistant can list option #2 as the first choice, so customers can get to it more quickly. If the pattern persists over time, it can change its behavior even more. Your assistant can return option #2 immediately, rather than asking a clarifying question.

As your assistant learns over time, your customers get the best answer more often and in fewer clicks.

### How autolearning works

Before your assistant can learn from customer behavior, it must observe a significant amount of real conversation data. The conversations take place in a channel such as the web chat, or in a custom application.

Logs of conversations and user decisions from your live environment are the data source for observation. Your assistant analyzes the logs to gain insights. (It doesn't watch real-time clicks during a conversation.)

When the assistant observes enough real conversation data from the live environment, it gains insights to help improve your assistant, providing a better customer experience.

When you publish your assistant to the live environment, autolearning starts training the assistant. To apply the autolearning improvements to your assistant's responses, [enable autolearning](#).

### Enabling autolearning

You can enable autolearning when the following conditions are met:

- **Ask clarifying question** is enabled in **Global settings**. For more information, see [Global settings for actions](#).
- You publish a version of your assistant to the live environment. For more information, see [Publishing your content](#).
- Your customers are interacting with a channel or custom application that is connected to the live environment.

To apply autolearning improvements to your assistant responses:

1. On the **Actions** page, click **Global settings** .
  2. Click the **Autolearning** tab.
  3. Set the **Autolearning in live environment** switch to **On**.
1. In the draft environment, you can preview your actions or your assistant. With autolearning set to **On**, preview in the draft environment uses the autolearning training from the live environment. For more information, see [Using Preview to test your action](#) or [Previewing your assistant](#).
  2. If you are happy with the results from testing the autolearning improvements in the draft environment, publish a new version of your assistant to the live environment to apply the improvements. For more information, see [Publishing your content](#).

## Learning from your data

Observations are made of only your customers' choices to improve only your assistant. These observations are not reused by IBM or shared in any way.

This observed user choices data is separate from the log data for which metrics are displayed on the **Analyze** page. The observation data is also separate from the information that is collected in all but Enterprise plan service instances and used by IBM for general service improvements. You can opt out of such use by specifying an opt-out header in your `/message` API requests. For more information, see [Opting out of log data use](#).

To prevent your own assistant from applying what it learns by observing user choices to your assistant, disable autolearning:

1. In **Global settings**, click the **Autolearning** tab.
2. Set the **Use autolearning to modify responses with training from live environment** switch to **Off**. This immediately disables the modifying of responses in the draft environment.
3. Publish a new version of your assistant to the live environment to completely disable the modifying of responses by autolearning.



**Note:** This feature is available only to users enrolled in the early access program. To gain access to early access features, complete the [Early access to beta features form](#).

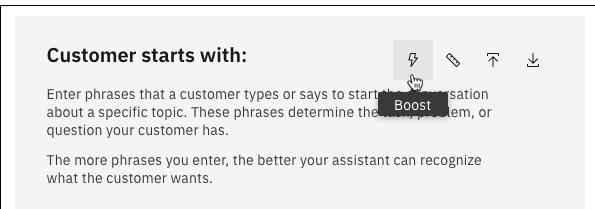
## Boost words

Use *boost words* to enter keywords or phrases to help the assistant recognize what a customer wants. If a customer's input includes matches the boost words or phrases, the assistant can be more confident to use an action.

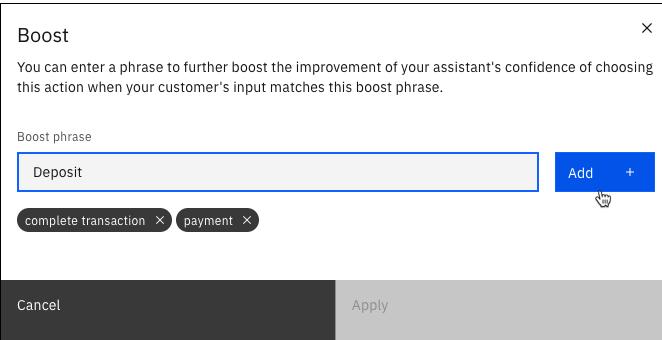
Boost words or phrases are used in addition to example phrases. They can only contain letters, underscores, hyphens, spaces, or dots. They must start with an alphanumeric character. Words or phrases can't be entered more than once per action, but you can use the same boost words or phrases across multiple actions.

To add boost words or phrases to an action:

1. Create or open an action.
2. In **Customer starts with**, ensure the action has at least one example phrase before adding any boost words.
3. Click the **Boost** icon.



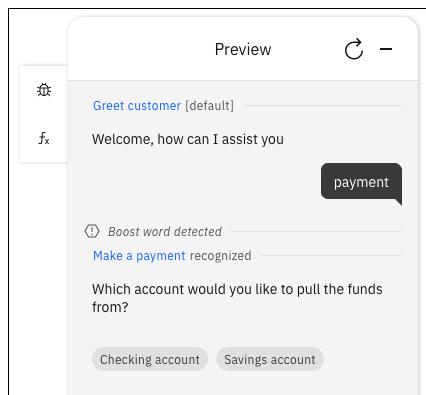
4. Enter a boost word or phrase, then click **Add**.



5. You can keep entering words or phrases one at a time. When you're finished, click **Apply**.

## Testing recognition of boost words

You can use **Preview** to test recognition of boost words. If a boost word or phrases matches, you can see **Boost word detected** in the Preview pane.



**Note:** This feature is available only to users enrolled in the early access program. To gain access to early access features, complete the [Early access to beta features form](#).

## Integrating with phone and Amazon Connect contact center

Connect your assistant to an Amazon Connect contact center with live agents.

Transfer customers from a chat with your assistant to live agents who can help them by phone. If customers ask to speak to someone, your assistant can forward them directly to customer support with the conversation history.

This integration creates a connection between your assistant and a contact center using Amazon Connect.

You need a Plus or Enterprise Plan to use this feature.

## Before you begin

You must have an Amazon Connect instance and phone numbers allocated for your contact center.

## Generate AWS access keys

Access keys are used for authentication and consist of two parts: an access key ID and a secret access key.

To generate AWS access keys to use with your assistant, see [AWS Account and Access Keys](#).

You cannot retrieve the secret key again after you complete the next step and **Save**. You must generate a new key if the current one is lost or forgotten.

## Set up the integration

To complete setup, you must have an assistant ready to deploy, your AWS access keys, and phone numbers allocated for this integration.

To integrate your assistant with Amazon Connect:

1. Click **Integrations**.

2. On the **Integrations** page, click "Add" on the **Phone** tile.

Select **Integrate with your contact center**. Click **Next**.

3. Select **Amazon Connect** on the **Set up a phone connection** page.

Click **Next**.

4. On the **Connect to your Amazon Connect account** page, specify the following values:

- the **Amazon Connect Instance ARN** from Amazon Connect. See [Find your Amazon Connect instance ID/ARN](#) to find your Amazon Connect ARN.
- the **Access Key ID**
- the **Access Key secret**

Click **Test connection** to verify the credentials.

- If **Invalid**, check your credentials and enter each again.

Click **Next**.

5. Copy the SIP address. You will need it for your Amazon Chime Voice Connector configuration. Click **Next**.

6. On the **Speech to Text** page, select the instance of the Speech to Text service you want to use.

- If you have an existing Speech to Text instances, select the instance you want to use from the list.
- If you do not have any existing Speech to Text instances, click **Create new instance** to create a new Plus or Enterprise instance.

7. In the **Choose your Speech to Text language model** field, select the language you want to use.

The list of language models is automatically filtered to use the same language as your assistant. To see all language models, toggle the **Filter models based on assistant language** switch to **Off**.

**Note:** If you created specialized custom models that you want your assistant to use, choose the Speech to Text

service instance that hosts the custom models now, and you can configure your assistant to use them later. The Speech to Text service instance must be hosted in the same location as your Watson Assistant service instance. For more information, see [Using a custom language model](#).

For more information about language models, see [Languages and models](#) in the Speech to Text documentation.

Click **Next**.

8. On the **Text to Speech** page, select the instance of the Text to Speech service you want to use.

- If you have an existing Text to Speech instances, select the instance you want to use from the list.
- If you do not have any existing Text to Speech instances, click **Create new instance** to create a new Standard instance.

1. Click **Save**.

The connection between your assistant and Amazon Connect is complete.

## Configure an Amazon Chime Voice Connector

After you create a Watson Assistant Phone integration, create an Amazon Chime Voice Connector.

1. Create an Amazon Chime Voice Connector under your AWS account. For more information, see [Creating an Amazon Chime Voice Connector](#).

2. Enable **Origination** settings to control inbound calling to your Watson Assistant instance.

In the **Inbound routes\*** section, click **New** to add a new inbound route. In the **Host** field enter the hostname you copied in the previous step when you created the Watson Assistant and Amazon Connect integration. In the **Port** field, enter 5061. Choose **TCP** from the **Protocol** drop-down list. Enter **1** in the **Priority** field and **5** in the **Weight** field. Click **Add**. Click **Save**.

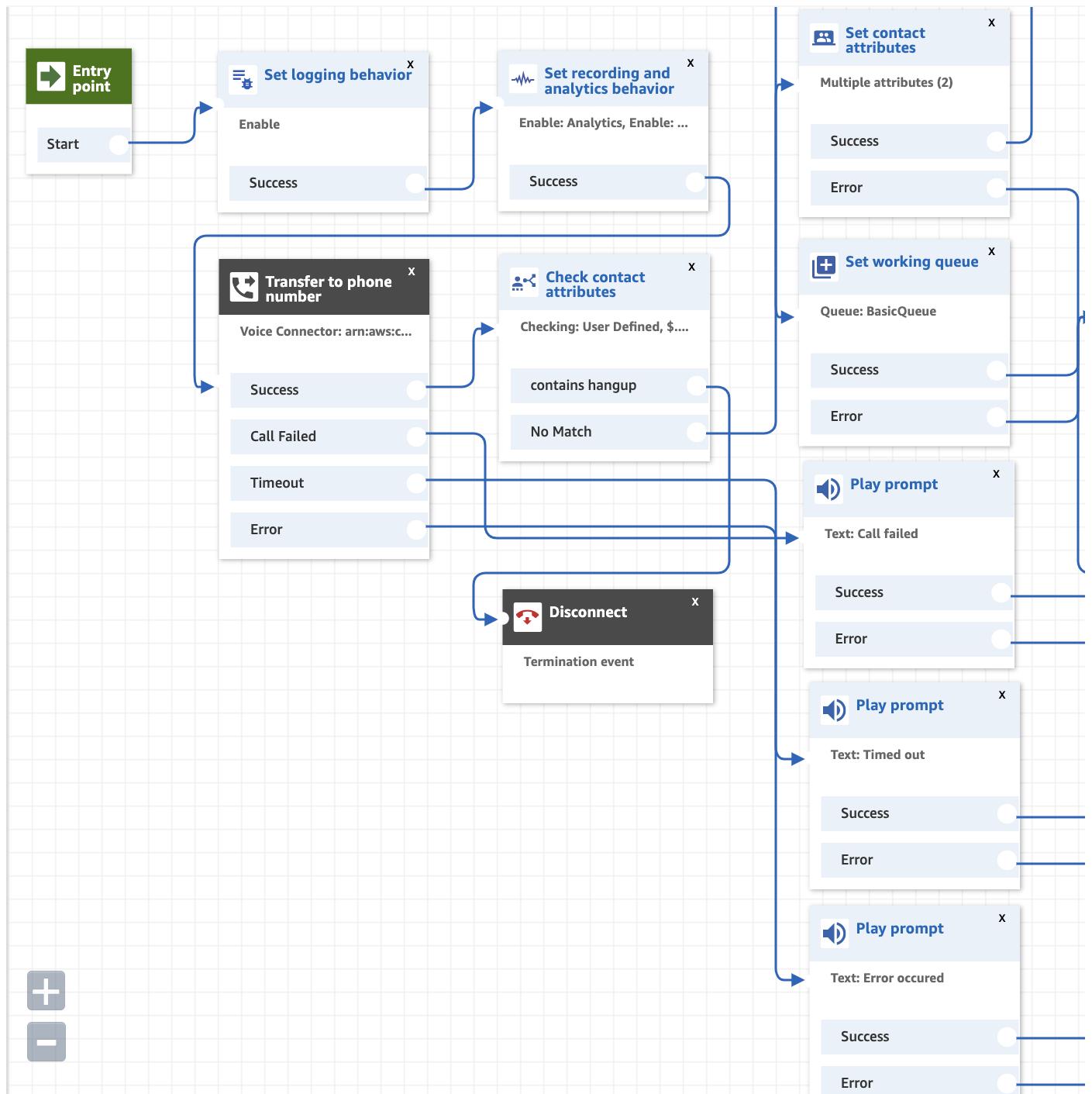
## Create an Amazon Connect Flow

A *flow* defines the customer experience with your contact center from start to finish.

You can find a sample flow in the [Watson Assistant Toolkit repository](#) that you can quickly set up and run a contact center. See [Import/export flows](#) for importing the sample flow.

However, you may want to create custom flows for your specific scenario. See [Create Amazon Connect Flows](#) for creating an Amazon Connect flow.

The following flow blocks and settings in the contact flow are necessary for integration to work properly.



## Connecting a caller to your assistant

Use the [Transfer to phone number](#) flow block to connect a caller to your Watson Assistant service instance.

1. In the **Transfer via** property, select **Voice Connector**. In the **Voice Connector ARN** property, select **Set manually** and enter the ARN of your Amazon Chime Voice Connector. To find your Voice Connector ARN: Open the [Amazon Chime console](#). Click **Voice connectors**. Select your Voice Connector and copy the value of **Voice Connector ARN**.
2. In the **To User** property, select **Use attribute**. From the **Type** drop-down list, select **System**. In the **Attribute** drop-down list, select **Dialed Number**.

3. In the **From User** property, select **Use attribute**. In the **Type** drop-down list select **System**. In the **Attribute** drop-down list, select **Customer Number**.

4. In the **User to User Information (UUID)** property, select **Use attribute**. From the **Type** drop-down list, select **System**. From the **Attribute** drop-down list, select **Instance ARN**.

5. Click **Save**.

Transfer to phone number X

Transfer a call to a phone number for voice interactions. [Learn more](#)

Transfer via

Phone number

Voice Connector

Voice Connector ARN

Set manually

[arn:aws:chime:us-east-1:xxx:vc/xxx](#)

Use attribute

To User

Set manually

Use attribute

Type

System

Attribute

Dialed Number

From User

Set manually

Use attribute

Type

System

Attribute

Customer Number

User to User Information (UUID)

Set manually

Use attribute

Type

System

Attribute

Instance ARN

Set connection timeout

Set timeout (in seconds)

60

Use attribute

## Decide whether to disconnect a call or transfer the caller to a live agent

Use the [Check contact attributes](#) flow block to branch the flow based on the **SessionHistoryKey** attribute.

1. In the **Attribute to check** property: From the **Type** drop-down list, select **User Defined**. In the **Attribute** field, type

`$.Attributes.SessionHistoryKey.`

2. In the **Conditions to check** property: Select `Contains` from the drop-down list and type `hangup` in the text field.

3. Click **Save**.

Check contact attributes

Branches based on a comparison to the value of a contact attribute. [Learn more](#)

Attribute to check

Type

User Defined

Attribute

`$.Attributes.SessionHistoryKey`

Conditions to check

x Contains ✓ hangup

No Match

[Add another condition](#)

Watson Assistant passes to the contact flow the `SessionHistoryKey` attribute which can be used for branching the contact flow based on a comparison to the value of this attribute. When the attribute is set to `hangup`, it's an indication to disconnect the flow. Otherwise, the caller should be transferred to a live agent.

## Displaying conversation history to a live agent

Use the [Set contact attributes](#) flow block, to make contact attributes accessible by the Contact Control Panel (CCP).

In the **Attribute to save** property, add the following attributes:

1. From the **Destination Type** drop-down list, select `User Defined`, and type `screenPopURL` in the **Destination Attribute** text field. Select `Use text`, type `https://web-chat.global.assistant.dev.watson.appdomain.cloud/loadAgentAppFrame.html?session_history_key` in the **Value** text box.
  2. From the **Destination Type** drop-down list, select `User Defined`, and type `sessionHistoryKey` in the **Destination Attribute** text field. From the **Type** drop-down list, select `User Defined`, type `$.Attributes.SessionHistoryKey` in the **Attribute** text box.
3. Click **Save**.

Attribute to save

Destination Type  
User Defined

Destination Attribute  
screenPopURL

Use text  
Value  
https://web-chat.global.assistant.dev.watson.app

Use attribute

Destination Type  
User Defined

Destination Attribute  
sessionHistoryKey

Use text

Use attribute

Type  
User Defined

Attribute  
\$.Attributes.SessionHistoryKey

[Add another attribute](#)

Watson Assistant passes to the contact flow the `SessionHistoryKey` contact attribute. The session history key can be used to retrieve conversation history and present it to a live agent.

This block configures your contact flow to provide a transcript of the assistant conversation to a live agent in a pop-out window. It helps the agent better understand and address a customer's needs.

## Transferring a caller to a live agent

Use the [Set working queue](#) flow block to specify the queue to be used when **Transfer to queue** is invoked. Use the [Transfer to queue](#) to place the caller in a queue.

## Transferring to a live agent when an error occurs

The phone integration disconnects the call if an error occurs during a conversation by sending a `SIP BYE` request. In that case, the `SessionHistoryKey` contact attribute is not passed to the contact flow.

## Claim a phone number for your Amazon Connect Contact Flow

To receive calls in your instance, you need to claim a phone number. If you did not claim a number when you created your Amazon Connect instance, follow [these steps](#) to claim one now. Attach the phone number to the contact flow you created for this integration.

## Complete your phone integration configuration

Open the Watson Assistant user interface and go to the phone integration you created earlier. On the **Phone number** page, enter the phone number you allocated for your Amazon Connect Contact Flow in the **Claim a phone number for your Amazon Connect Contact Flow** step.

## Adding transfer support to your assistant

Configure your assistant to transfer calls to a live agent using the *Connect To Agent* response type. For instructions, see [Transferring a call to a live agent](#).

Use the following format:

```
$ {
  "generic": [
    {
      "response_type": "connect_to_agent",
      "transfer_info": {
        "target": {
          "amazon_connect": {
            "custom_data": {
              "contact_attribute1": "test"
            }
          }
        }
      },
      "agent_available": {
        "message": "Ok, I'm transferring you to an agent."
      },
      "agent_unavailable": {
        "message": "Agent is unavailable."
      }
    }
  ]
}
```

Parameters listed in the `custom_data` object are sent to the Amazon Connect contact flow using the [API\\_UpdateContactAttributes](#) REST API.

# Glossary

Term	Definition
Action	Actions represent the tasks or questions that your assistant can help customers with. Each action has a beginning and an end, making up a conversation between the assistant and a customer. <a href="#">Learn more</a> .
Ask clarifying question	A feature that enables the assistant to ask customers to clarify (disambiguate) their meaning when the assistant isn't sure what a user wants to do next. <a href="#">Learn more</a> .
Assistant	Container for your actions, channels, and integrations. You add actions and at least one channel to an assistant, and then deploy the assistant when you are ready to start helping your customers. <a href="#">Learn more</a> .
Change conversation topic	A feature that gives the user the power to direct the conversation. It allows digressions and prevents customers from getting stuck in a dialog thread; they can switch topics whenever they choose. <a href="#">Learn more</a> .
Channel	The location where your assistant interacts with your users, for example, over the phone, on a website, or in Slack. At least one channel is required for every assistant. <a href="#">Learn more</a> .
Completion	Measures how often within a given time period users reach the end step of an action. <a href="#">Learn more</a> .
Content	The conversation logic and words that are used to respond to your customer. Content is required for every assistant. <a href="#">Learn more</a> .
Environment	You can group your work in separate containers that are called <i>environments</i> . Each environment contains its own content, channels, and extensions. Environments also have their own IDs, URLs, and service credentials that can be referenced by external services. Each new assistant comes with two environments: the draft environment and the live environment. Your customers interact with assistants on the live environment and cannot interact with assistants on the draft environment. The separation of these two environments allows you to build and iterate on your content separately from what your customers see. You do not want customers to stumble upon an incomplete action that leads them to a dead end. For Enterprise plans, you can add up to three environments as a staging area to test your assistant before deployment. You can build content in the draft environment and test versions of your content in the extra environments. <a href="#">Learn more</a> .
Escalation	If your assistant is integrated with one of the supported service desk systems, you can build in logic that transfers, or escalates, the conversation to a human when necessary. <a href="#">Learn more</a> .
Incompletion	Reasons why an action is not completed by a user, including escalated to agent, started a new action, stuck on a step, or abandoned or ongoing. <a href="#">Learn more</a>
Integrations	Add-ons to the end experience that help solve specific user problems, for example, connecting to a human agent or searching existing help content. Integrations are not required for an assistant, but they are recommended. <a href="#">Learn more</a> .
Message	A single turn within a conversation that includes a single call to the /message API endpoint and its corresponding response.
Monthly active user (MAU)	A single unique user who interacts with an assistant one or many times in a given month. <a href="#">Learn more</a> .
Preview	Embeds your assistant in a chat window that is displayed on an IBM-branded web page. From the preview, you can test how a conversation flows through your assistant, from end to end. <a href="#">Learn more</a> .
Recognition	Measurement of how many requests are being recognized by the assistant and routed into starting an action. <a href="#">Learn more</a> .

Response	To create your assistant's response in an action step, you use the <code>Assistant says</code> section. This represents the text or speech the assistant delivers to a user at a particular step. Depending on the step, you can add a complete answer to a user's question or ask a follow-up question. <a href="#">Learn more</a> .
Step	A step that you add to an action represents a single interaction or exchange of information with a customer, a turn in the conversation. <a href="#">Learn more</a> .
Training	To set up a Watson instance with components that enable the system to function in a particular domain (for example: corpus content, training data that generates machine learning models, programmatic algorithms, annotators, or other ground truth components) and then making improvements and updates to these components based on accuracy analysis.
Variable	A variable is data that a customer shares with the assistant, which is collected and saved so it can be referenced later. In actions, you can collect <code>action</code> and <code>session</code> variables. <a href="#">Learn more</a> .
Web chat	A channel that you can use to embed your assistant in your company website. <a href="#">Learn more</a> .
Webhook	A mechanism for calling out to an external program during a conversation. For example, your assistant can call an external service to translate a string from English to French and back again in the course of the conversation. <a href="#">Learn more</a> .

## Glossary

# Response types reference

You can use the JSON editor to specify responses of many different types.

For more information about using response types in the JSON editor, see [Defining responses using the JSON editor](#).

The following response types are supported in the JSON editor.

audio

Plays an audio clip specified by a URL.

## Integration channel support

Web chat	Phone	SMS	Slack	Facebook	WhatsApp
✓	✓	✓	✓	✓	✓

- Some channel integrations do not display audio titles or descriptions.

## Fields

Name	Type	Description	Required?
response_type	string	audio	Y
source	string	The https: URL of the audio clip. The URL can specify either an audio file or an audio clip on a supported hosting service.	Y
title	string	The title to show before the audio player.	N
description	string	The text of the description that accompanies the audio player.	N
alt_text	string	Descriptive text that can be used for screen readers or other situations where the audio player cannot be seen.	N
channel_options.voice_telephony.loop	string	Whether the audio clip should repeat indefinitely (phone integration only).	N

The URL specified by the `source` property can be either of the following:

- The URL of an audio file in any standard format such as MP3 or WAV. In the web chat, the linked audio clip will render as an embedded audio player.
- The URL of an audio clip on a supported streaming service. In the web chat, the linked audio clip will render using the embeddable player for the hosting service.

Specify the URL you would use to access the audio file in your browser (for example,

<https://soundcloud.com/ibmresearch/fallen-star-amped>). You do not need to convert the URL to an embeddable form; the web chat will do this automatically.

You can embed audio hosted on the following services:

- SoundCloud

- o [Mixcloud](#)

**Note:** For the phone integration, the URL must specify an audio file that is single-channel (mono) and PCM-encoded, and is sampled at 8,000 Hz with 16 bits per sample.

## Example

This example plays an audio clip with a title and descriptive text.

```
{
  "generic": [
    {
      "response_type": "audio",
      "source": "https://example.com/audio/example-file.mp3",
      "title": "Example audio file",
      "description": "An example audio clip returned as part of a multimedia response."
    }
  ]
}
```

channel\_transfer

Requests that the conversation be transferred to a different channel integration. Currently, the web chat integration is the only supported target of a channel transfer.

## Integration channel support

Phone	SMS	Slack	Facebook	WhatsApp
✓	✓	✓	✓	✓

- The indicated channel integrations support *initiating* a channel transfer (currently, the web chat integration is the only supported transfer target).
- Initiating a channel transfer from the phone integration requires that the SMS integration also be configured.

## Fields

Name	Type	Description	Required?
response_type	string	channel_transfer	Y
message_to_user	string	A message to display to the user before the link for initiating the transfer.	Y
transfer_info	object	Information used by an integration to transfer the conversation to a different channel.	Y
transfer_info.target.chat	string	The URL for the website hosting the web chat to which the conversation is to be transferred.	Y

## Example

This example requests a transfer from WhatsApp to the web chat. In addition to the `channel_transfer` response, the output also includes a `text` response to be displayed by the web chat integration after the transfer. The use of the `channels` array ensures that the `channel_transfer` response is handled only by the WhatsApp integration (before the transfer), and the `connect_to_agent` response only by the web chat integration (after the transfer). For more information about using `channels`

to target specific integrations, see [Targeting specific integrations](#).

```
{  
  "generic": [  
    {  
      "response_type": "channel_transfer",  
      "channels": [  
        {  
          "channel": "whatsapp"  
        }  
      ],  
      "message_to_user": "Click the link to connect with an agent using our website.",  
      "transfer_info": {  
        "target": {  
          "chat": {  
            "url": "https://example.com/webchat"  
          }  
        }  
      }  
    },  
    {  
      "response_type": "connect_to_agent",  
      "channels": [  
        {  
          "channel": "chat"  
        }  
      ],  
      "message_to_human_agent": "User asked to speak to an agent.",  
      "agent_available": {  
        "message": "Please wait while I connect you to an agent."  
      },  
      "agent_unavailable": {  
        "message": "I'm sorry, but no agents are online at the moment. Please try again later."  
      },  
      "transfer_info": {  
        "target": {  
          "zendesk": {  
            "department": "Payments department"  
          }  
        }  
      }  
    }  
  ]  
}
```

#### connect\_to\_agent

Requests that the conversation be transferred to a live agent for help. Service desk support must be configured for the channel integration.

## Integration channel support

Web chat	Phone	WhatsApp
✓	✓	✓

- For information about adding service desk support to the web chat integration, see [Adding contact center support](#).
- For information about adding service desk support to the phone integration, see [Configuring backup call center support](#).

## Fields

Name	Type	Description	Required?
------	------	-------------	-----------

response_type	string	connect_to_agent	Y
message_to_human_agent	string	A message to display to the live agent to whom the conversation is being transferred.	Y
agent_available	string	A message to display to the user when agents are available.	Y
agent_unavailable	string	A message to display to the user when no agents are available.	Y
transfer_info	object	Information used by the web chat service desk integrations for routing the transfer.	N
transfer_info.target.zendesk.department	string	A valid department from your Zendesk account.	N
transfer_info.target.salesforce.button_id	string	A valid button ID from your Salesforce deployment.	N

## Example

This example requests a transfer to a live agent and specifies messages to be displayed both to the user and to the agent at the time of transfer.

```
{
  "generic": [
    {
      "response_type": "connect_to_agent",
      "message_to_human_agent": "User asked to speak to an agent.",
      "agent_available": {
        "message": "Please wait while I connect you to an agent."
      },
      "agent_unavailable": {
        "message": "I'm sorry, but no agents are online at the moment. Please try again later."
      }
    }
  ]
}
```

## date

Displays an interactive date picker the customer can use to specify a date value.

## Integration channel support

### Web chat



- In the web chat, the customer can specify a date value either by clicking the interactive date picker or typing a date value in the input field.

## Fields

Name	Type	Description	Required?
response_type	string	date	Y

## Example

This example sends a text response asking the user to specify a date, and then shows an interactive date picker.

```
{  
  "generic": [  
    {  
      "response_type": "text",  
      "text": "What day will you be checking in?"  
    },  
    {  
      "response_type": "date"  
    }  
  ]  
}
```

dtmf

Sends commands to the phone integration to control input or output using dual-tone multi-frequency (DTMF) signals. (DTMF is a protocol used to transmit the tones that are generated when a user presses keys on a push-button phone.)

## Integration channel support

### Phone



### Fields

Name	Type	Description	Required?
response_type	string	dtmf	Y
command_info	object	Information specifying the DTMF command to send to the phone integration.	Y
command_info.type	string	The DTMF command to send (collect, disable_barge_in, enable_barge_in, or send).	Y
command_info.parameters	object	See <a href="#">Handling phone interactions</a>	N

The `command_info.type` field can specify any of the following supported commands:

- `collect`: Collects DTMF keypad input.
- `disable_barge_in`: Disables DTMF barge-in so that playback from the phone integration is not interrupted when the customer presses a key.
- `enable_barge_in`: Enables DTMF barge-in so that the customer can interrupt playback from the phone integration by pressing a key.
- `send`: Sends DTMF signals.

For detailed information about how to use each of these commands, see [Handling phone interactions](#).

## Example

This example shows the `dtmf` response type with the `collect` command, used to collect DTMF input. For more information, including examples of other DTMF commands, see [Handling phone interactions](#).

```
{
  "generic": [
    {
      "response_type": "dtmf",
      "command_info": {
        "type": "collect",
        "parameters": {
          "termination_key": "#",
          "count": 16,
          "ignore_speech": true
        }
      },
      "channels": [
        {
          "channel": "voice_telephony"
        }
      ]
    }
  ]
}
```

end\_session

Sends a command to the channel ending the session. This response type instructs the phone integration to hang up the call.

## Integration channel support

Phone

SMS



- The SMS integration supports ending a session by using the `terminateSession` action command.

## Fields

Name	Type	Description	Required?
response_type	string	end_session	Y

For the phone integration, you can use the `channel_options` object to include custom headers with the SIP `BYE` request that is generated. For more information, see [End the call](#).

## Example

This example uses the `end_session` response type to end a conversation.

```
{
  "generic": [
    {
      "response_type": "end_session"
    }
  ]
}
```

iframe

Embeds content from an external website as an HTML `iframe` element.

## Integration channel support

## Web chat

## Facebook



- Currently, the web chat integration ignores the `description` and `image_url` properties. Instead, the description and preview image are dynamically retrieved from the source at run time.

## Fields

Name	Type	Description	Required?
<code>response_type</code>	string	<code>iframe</code>	Y
<code>source</code>	string	The URL of the external content. The URL must specify content that is embeddable in an HTML <code>iframe</code> element.	Y
<code>title</code>	string	The title to show before the embedded content.	N
<code>description</code>	string	The text of the description that accompanies the embedded content.	N
<code>image_url</code>	string	The URL of an image that shows a preview of the embedded content.	N

Note that different sites have varying restrictions for embedding content, and different processes for generating embeddable URLs. An embeddable URL is one that can be specified as the value of the `src` attribute of the `iframe` element.

For example, to embed an interactive map using Google Maps, you can use the Google Maps Embed API. (For more information, see [The Maps Embed API overview](#).) Other sites have different processes for creating embeddable content.

For technical details about using `Content-Security-Policy: frame-src` to allow embedding of your website content, see [CSP: frame-src](#).

## Example

This example embeds an `iframe` with a title and description.

```
{  
  "generic": [  
    {  
      "response_type": "iframe",  
      "source": "https://example.com/embeddable/example",  
      "title": "Example iframe",  
      "description": "An example of embeddable content returned as an iframe response."  
    }  
  ]  
}
```

`image`

Displays an image specified by a URL.

## Integration channel support

Web chat	SMS	Slack	Facebook	WhatsApp
✓	✓	✓	✓	✓

- Some channel integrations do not display image titles or descriptions.

## Fields

Name	Type	Description	Required?
response_type	string	image	Y
source	string	The https: URL of the image. The specified image must be in .jpg, .gif, or .png format.	Y
title	string	The title to show before the image.	N
description	string	The text of the description that accompanies the image.	N
alt_text	string	Descriptive text that can be used for screen readers or other situations where the image cannot be seen.	N

## Example

This example displays an image with a title and descriptive text.

```
{  
  "generic": [  
    {  
      "response_type": "image",  
      "source": "https://example.com/image.jpg",  
      "title": "Example image",  
      "description": "An example image returned as part of a multimedia response."  
    }  
  ]  
}
```

option

Presents a set of options (such as buttons or a drop-down list) that users can choose from. The selected value is then sent to the assistant as user input. An `options` response is automatically defined when you choose the **Options** customer response type for a step (for more information, see [Collecting information from your customers](#)).

## Integration channel support

Web chat	Phone	SMS	Slack	Facebook	WhatsApp
✓	✓	✓	✓	✓	✓

- The way in which options are presented varies depending on the channel integration. The `preference` field is supported when possible, but not all channels support drop-down lists or buttons.

## Fields

Name	Type	Description	Required?
response_type	string	option	Y
title	string	The title to show before the options.	Y
description	string	The text of the description that accompanies the options.	N

preference	string	The preferred type of control to display, if supported by the channel (dropdown or button).	N
options	list	A list of key/value pairs specifying the options from which the user can choose.	Y
options[].label	string	The user-facing label for the option.	Y
options[].value	object	An object defining the response that will be sent to the Watson Assistant service if the user selects the option.	Y
options[].value.input	object	An object that includes the message input corresponding to the option, including input text and any other field that is a valid part of a Watson Assistant message. For more information about the structure of message input, see the <a href="#">API Reference</a> .	N

## Example

This example presents two options (`Buy something` and `Exit`).

```
{
  "generic": [
    {
      "response_type": "option",
      "title": "Choose from the following options:",
      "preference": "button",
      "options": [
        {
          "label": "Buy something",
          "value": {
            "input": {
              "text": "Place order"
            }
          }
        },
        {
          "label": "Exit",
          "value": {
            "input": {
              "text": "Exit"
            }
          }
        }
      ]
    }
  ]
}
```

pause

Pauses before sending the next message to the channel, and optionally sends a "user is typing" event (for channels that support it).

## Integration channel support

Web chat	Facebook	WhatsApp
✓	✓	✓

- With the phone integration, you can add a pause by including the SSML `break` element in the assistant output. For more information, see the [Text to Speech documentation](#).

## Fields

Name	Type	Description	Required?
response_type	string	pause	Y
time	int	How long to pause, in milliseconds.	Y
typing	boolean	Whether to send the "user is typing" event during the pause. Ignored if the channel does not support this event.	N

## Example

This example sends the "user is typing" event while pausing for 5 seconds.

```
{  
  "output": {  
    "generic": [  
      {  
        "response_type": "pause",  
        "time": 5000,  
        "typing": true  
      }  
    ]  
  }  
}
```

speech\_to\_text

Sends a command to the Speech to Text service instance used by the phone integration. These commands can dynamically change the configuration or behavior of the service during a conversation.

## Integration channel support

### Phone



## Fields

Name	Type	Description	Required?
response_type	string	speech_to_text	Y
command_info	object	Information specifying the command to send to the Speech to Text.	Y
command_info.type	string	The command to send (currently only the <code>configure</code> command is supported).	Y
command_info.parameters	object	See <a href="#">Applying advanced settings to the Speech to Text service</a>	N

The `command_info.type` field can specify any of the following supported commands:

- `configure`: Dynamically updates the Speech to Text configuration. Configuration changes can be applied only to the next conversation turn, or for the rest of the session.

For detailed information about how to this command, see [Applying advanced settings to the Speech to Text service](#).

## Example

This example uses the `speech_to_text` response type with the `configure` command to change the language model used by the Speech to Text service to Spanish, and to enable smart formatting.

```
{  
  "generic": [  
    {  
      "response_type": "speech_to_text",  
      "command_info": {  
        "type": "configure",  
        "parameters": {  
          "narrowband_recognize": {  
            "model": "es-ES_NarrowbandModel",  
            "smart_formatting": true  
          }  
        }  
      }  
    },  
    "channels": [  
      {  
        "channel": "voice_telephony"  
      }  
    ]  
  ]  
}
```

## start\_activities

Sends a command to a channel integration to start one or more activities that are specific to that channel. You can use this response type to restart any activity you previously stopped using the `stop_activities` response type.

## Integration channel support

### Phone



### Fields

Name	Type	Description	Required?
response_type	string	<code>start_activities</code>	Y
activities	list	A list of objects identifying the activities to start.	Y
activities[].type	string	The name of the activity to start.	Y

Currently, the following activities for the phone integration can be started:

- `speech_to_text_recognition`: Starts recognizing speech. Streaming audio to the Speech to Text service is resumed.
- `dtmf_collection`: Starts processing of inbound DTMF signals.

## Example

This example uses the `start_activities` response type to restart recognizing speech. Because this command is specific to the phone integration, the `channels` property specifies `voice_telephony` only.

```
{
```

```

"generic": [
  {
    "response_type": "start_activities",
    "activities": [
      {
        "type": "speech_to_text_recognition"
      }
    ],
    "channels": [
      {
        "channel": "voice_telephony"
      }
    ]
  }
]

```

## stop\_activities

---

Sends a command to a channel integration to stop one or more activities that are specific to that channel. The activities remain stopped until they are restarted using the `start_activities` response type.

## Integration channel support

### Phone



## Fields

Name	Type	Description	Required?
response_type	string	stop_activities	Y
activities	list	A list of objects identifying the activities to stop.	Y
activities[].type	string	The name of the activity to stop.	Y

Currently, the following activities for the phone integration can be stopped:

- `speech_to_text_recognition`: Stops recognizing speech. All streaming audio to the Speech to Text service is stopped.
- `dtmf_collection`: Stops processing of inbound DTMF signals.

## Example

This example uses the `stop_activities` response type to stop recognizing speech. Because this command is specific to the phone integration, the `channels` property specifies `voice_telephony` only.

```

{
  "generic": [
    {
      "response_type": "stop_activities",
      "activities": [
        {
          "type": "speech_to_text_recognition"
        }
      ],
      "channels": [
        {
          "channel": "voice_telephony"
        }
      ]
    }
  ]
}

```

```
        }
    ]
}
}
```

## text

Displays text (or reads it aloud, for the phone integration). To add variety, you can specify multiple alternative text responses. If you specify multiple responses, you can choose to rotate sequentially through the list, choose a response randomly, or output all specified responses.

## Integration channel support

Web chat	Phone	SMS	Slack	Facebook	WhatsApp
✓	✓	✓	✓	✓	✓

## Fields

Name	Type	Description	Required?
response_type	string	text	Y
values	list	A list of one or more objects defining text response.	Y
values.[n].text_expression	object	An object describing the text of the response.	N
values.[n].text_expression.concat	list	A list of objects that form components of the text response. These objects can include scalar text strings and references to variables.	N
selection_policy	string	How a response is selected from the list, if more than one response is specified. The possible values are sequential, random, and multiline.	N
delimiter	string	The delimiter to output as a separator between responses. Used only when selection_policy=multiline. The default delimiter is newline ( ).	N

## Example

This examples displays a greeting message to the user.

```
{
  "generic": [
    {
      "response_type": "text",
      "values": [
        {
          "text_expression": {
            "concat": [
              {
                "scalar": "Hi, "
              },
              {
                "variable": "step_472"
              },
              {
                "scalar": ". How can I help you?"
              }
            ]
          }
        }
      ]
    }
  ]
}
```

```

        }
    ],
    "selection_policy": "sequential"
}
]
}

```

## text\_to\_speech

---

Sends a command to the Text to Speech service instance used by the phone integration. These commands can dynamically change the configuration or behavior of the service during a conversation.

### Integration channel support

#### Phone



### Fields

Name	Type	Description	Required?
response_type	string	text_to_speech	Y
command_info	object	Information specifying the command to send to the Text to Speech.	Y
command_info.type	string	The command to send (configure, disable_barge_in, or enable_barge_in).	Y
command_info.parameters	object	See <a href="#">Applying advanced settings to the Text to Speech service</a>	N

The `command_info.type` field can specify any of the following supported commands:

- `configure`: Dynamically updates the Text to Speech configuration. Configuration changes can be applied only to the next conversation turn, or for the rest of the session.
- `disable_barge_in`: Disables speech barge-in so that playback from the phone integration is not interrupted when the customer speaks.
- `enable_barge_in`: Enables speech barge-in so that the customer can interrupt playback from the phone integration by speaking.

For detailed information about how to use each of these commands, see [Applying advanced settings to the Text to Speech service](#).

### Example

This example uses the `text_to_speech` response type with the `configure` command to change the voice used by the Text to Speech service.

```
{
  "generic": [
    {
      "response_type": "text_to_speech",
      "command_info": {
        "type": "configure",
        "parameters": {

```

```

        "synthesize": {
            "voice": "en-US_LisaVoice"
        }
    },
    "channels": [
        {
            "channel": "voice_telephony"
        }
    ]
}

```

#### user\_defined

---

A custom response type containing any JSON data the client or integration knows how to handle. For example, you might customize the web chat to display a special kind of card, or build a custom application to format responses using a table or chart.



**Note:** The user-defined response type is not displayed unless the channel has code to handle it. For more information about customizing the web chat, see [Applying advanced customizations](#).

## Integration channel support

Web chat	Phone	SMS	Slack	Facebook	WhatsApp
✓	✓*	✓*	✓	✓	✓

- With the phone integration, the `user_defined` response type is used to send legacy commands (for example, `vgwActForceNoInputTurn` or `vgwActSendSMS`). For more information, see [Handling phone interactions](#).
- With the SMS integration, the `user_defined` response type is used to send action commands (for example, `terminateSession` or `smsActSendMedia`).

## Fields

Name	Type	Description	Required?
<code>response_type</code>	string	<code>user_defined</code>	Y
<code>user_defined</code>	object	An object containing any data the client or integration knows how to handle. This object can contain any valid JSON data, but it cannot exceed a total size of 5000 bytes.	Y

## Example

This examples shows a generic example of a user-defined response. The `user_defined` object can contain any valid JSON data.

```

{
  "generic": [
    {
      "response_type": "user_defined",
      "user_defined": {
        "field_1": "String value",
        "array_1": [
          1,
          2
        ],
        "object_1": {
          "key_1": "value_1"
        }
      }
    }
  ]
}

```

```

        "property_1": "Another string value"
    }
}
]
}

```

## video

Displays a video specified by a URL.

### Integration channel support

Web chat	SMS	Slack	Facebook	WhatsApp
✓	✓	✓	✓	✓

- Some channel integrations do not video titles or descriptions.

### Fields

Name	Type	Description	Required?
response_type	string	video	Y
source	string	The <code>https:</code> URL of the video. The URL can specify either a video file or a streaming video on a supported hosting service.	Y
title	string	The title to show before the video.	N
description	string	The text of the description that accompanies the video.	N
alt_text	string	Descriptive text that can be used for screen readers or other situations where the video cannot be seen.	N
channel_options.chat.dimensions.base_height	string	The base height (in pixels) to use to scale the video to a specific display size.	N

The URL specified by the `source` property can be either of the following:

- The URL of a video file in a standard format such as MPEG or AVI. In the web chat, the linked video will render as an embeddable video player.
- HLS (`.m3u8`) and DASH (MPD) streaming videos are not supported.
- The URL of a video hosted on a supported video hosting service. In the web chat, the linked video will render using the embeddable player for the hosting service.

Specify the URL you would use to view the video in your browser (for example, `https://www.youtube.com/watch?v=52bpMKVigGU`). You do not need to convert the URL to an embeddable form; the web chat will do this automatically.

You can embed videos hosted on the following services:

- [YouTube](#)
- [Facebook](#)

- [Vimeo](#)
- [Twitch](#)
- [Streamable](#)
- [Wistia](#)
- [Vidyard](#)

## Example

This example displays an video with a title and descriptive text.

```
{  
  "generic": [  
    {  
      "response_type": "video",  
      "source": "https://example.com/videos/example-video.mp4",  
      "title": "Example video",  
      "description": "An example video returned as part of a multimedia response."  
    }  
  ]  
}
```

# Preview runtime errors

You may see these system errors when testing your assistant in [Preview](#).

 **Note:** Some of these errors correspond to advanced development tasks.

Message	Description
No actions to interpret	Assistant has no content.
No action triggered	Error when an action is not found, for example, when one action specifies another action that isn't defined yet.
Conversation ended. Maximum of %d actions visited.	Limit is 20 per conversation.
Maximum number of step iterations for a single action	When a particular step is reached more than 50 times within an action.
Error when updating output in [%s]. The output is [%s]	When there was an error in updating the output, for example, when using a SpEL expression.
Error in context update	Error when setting a variable to an invalid ASEL expression.
Error when updating context with context of step %s. Step context is [%s].	Error when a runtime exception occurs while setting a context, for example, division by zero.
Multiple actions have the same title: %s	Actions need to have unique names.
Action %s is invalid because step order includes a cycle. Evaluation could result in an infinite loop. Check &quot;next_step&quot; property of each step in the action.	Step could get caught in an infinite loop and not end.
Expression evaluation error inside condition of %s. The syntax is valid, but cannot be evaluated. Condition defaulted to false. Check that objects in expression are not null or out of bounds. Error: %s	When an unexpected error occurs while checking a condition, for example, division by 0.
Provided expression [%s] cannot be evaluated as a number	ASEL error message
Provided expression [%s] cannot be evaluated as Boolean	ASEL error message
Provided expression [%s] cannot be evaluated as a string	ASEL error message
Provided expression [%s] cannot be evaluated	ASEL error message
Actions array contains an element that is not a JsonObject. Cannot build an internal representation.	Advanced programming message

Actions array contains elements with the same id (attribute 'action'). Cannot build an internal representation.

Advanced programming message

No target step %s found for jump-to resolver

Advanced programming message

For resolver of type jump\_to the target step cannot be the same as the source step.

Advanced programming message

# Built-in global variables

By writing expressions, you can access a set of global system variables that provide information about the conversation.

Each variable contains a JSON object that is taken from the `message` method input or output. For more information about these objects, see the [API Reference](#) information for the `message` method request and response.

**⚠ Important:** These variables are special system objects that require syntax different from the standard variable notation. To reference any of these values in an expression, use the variable name by itself (not including `?` or `{}` characters).

Variable	Description	Expression example
<code>entities[]</code>	The <code>output.entities</code> array from the most recent <code>message</code> response.	<code>entities[0].value</code>
<code>intents[]</code>	The <code>output.intents</code> array from the most recent <code>message</code> response.	<code>intents[0].confidence</code>
<code>input</code>	The <code>input</code> object from the most recent <code>message</code> request sent to the assistant.	<code>input.text</code>
<code>output</code>	The <code>output</code> object from the most recent <code>message</code> response. Currently, only <code>output.debug</code> is included.	<code>output.debug.turn_events[0]</code>

**Built-in global system variables**

# Integration variables

By writing expressions, you can access integration variables, which are context variables that are specific to integrations (such as the web chat and phone integrations). All integration variables are contained in the `system_integrations` JSON object, which you can access using the following expression:

```
 ${system_integrations}
```

The `system_integrations` object has the following structure:

```
{  
  "channel": {  
    "name": "{channel_name}",  
    "private": {  
      "user": {  
        "id": "{user_id}",  
        "phone_number": "{phone_number}",  
        "name": "{name}"  
      }  
    }  
  },  
  "chat": {...},  
  "voice_telephony": {...},  
  "text_messaging": {...},  
  "whatsapp": {...},  
  "slack": {...},  
  "facebook": {...},  
  "teams": {...}  
}
```

channel

Always included. This object contains general information about the channel that is being used to communicate with the assistant.

## Properties



**Note:** Properties contained in the `private` object are treated as private variables, which are not included in logs.

Name	Type	Description
<code>channel.name</code>	String	The name of the channel that is in use. One of the following values: <ul style="list-style-type: none"><li>• Web chat</li><li>• Phone</li><li>• SMS</li><li>• Whatsapp</li><li>• Slack</li><li>• Facebook Messenger</li></ul>
<code>channel.private.user.id</code>	String	The ID of the user who is interacting with the assistant through the channel. This ID is specific to the channel and might be different from the user ID Watson Assistant uses for billing purposes. For more information, see <a href="#">Channel user IDs</a> .

channel.private.user.phone_number	String	The phone number associated with the user. Set by the phone, SMS, and WhatsApp integrations.
channel.private.user.name	String	The name of the user who is interacting with the assistant through the channel. Set by the Microsoft Teams integration.

#### Properties of the channel object

## Example JSON

```
"channel": {
  {
    "name": "Web chat",
    "private": {
      "user": {
        "id": "anonymous_IBMuid-727c0302-6fd7-4abb-b7ee-f06b4bf30e99"
      }
    }
  }
}
```

## Channel user ID

The `channel.private.user.id` property, which is set by the channel integration, specifies a user ID for the customer that is specific to the channel. The source of this user ID depends on the channel:

Channel	Channel user ID
Web chat	The user ID set by the web chat instance. For more information, see <a href="#">Managing user identity information</a> .
Slack	The Slack member ID (for example, U2147483697).
Facebook	The Facebook sender ID (for example, 4310101122439797).
Whatsapp	The customer's phone number.
Phone	The customer's phone number.
SMS with Twilio	The customer's phone number.

#### Sources of the channel user ID

chat\_\_\_\_\_

Included only if the web chat integration is in use.

## Properties

Name	Type	Description
browser_info.browser_name	String	The browser name, such as <code>chrome</code> , <code>edge</code> , or <code>firefox</code> .
browser_info.browser_version	String	The browser version, such as <code>109.0.0</code> .

browser_info.browser_OS	String	The operating system of the customer's computer, such as Mac OS.
browser_info.language	String	The default locale code of the browser, such as en-US.
browser_info.page_url	String	The URL of the web page where the web chat is embedded, not including any query parameters or hashes.
browser_info.screen_resolution	String	The height and width of the browser window, such as width: 1440, height: 900.
browser_info.user_agent	String	The content of the HTTP User-Agent request header.
browser_info.client_ip_address	String	The IP address of the customer's computer.
browser_info.ip_address_list	Array	An array IP addresses specified by HTTP X-Forwarded-For request headers.

#### Properties of the chat object

### Example JSON

```

"chat": {
  "browser_info": {
    "browser_name": "chrome",
    "browser_version": "109.0.0",
    "browser_OS": "Mac OS",
    "language": "en-US",
    "page_url": "https://us-south.assistant.watson.cloud.ibm.com/crn%3Av1%3Abuemix%3Apublic%3Aconversation%3Aus-south%3Aa%2Fc41400d63d91741a749091dc63574c2c%3Ab696c1e5-7316-4fb0-a61c-664438397e91%3A%3A/assistants/e344fcfe-506c-449f-a182-ebdefe4356ad/actions/actions/custom/edit/action_28584",
    "screen_resolution": "width: 1920, height: 1080",
    "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36",
    "client_ip_address": "65.191.135.254",
    "ip_address_list": [
      "65.191.135.254",
      "104.99.56.143",
      "10.185.27.136",
      "172.30.226.64"
    ]
  }
},
"channel": {
  "name": "Web chat",
  "private": {
    "user": {
      "id": "anonymous_IBMid-727c0302-6fd7-4abb-b7ee-f06b4bf30e99"
    }
  }
}
}

```

### Example expressions

- This expression tests whether the customer is using the Chrome browser:

```
$ ${system_integrations}.chat.browser_info.browser_name.equals("chrome")
```

You might specify this expression as the value for a boolean session variable, which you could then use in a step condition.

For example, if your website supports only Chrome, you could have a step conditioned on this variable being `false` and has the following output: `I see you aren't using the Chrome browser. Some features of our website work only on Chrome.`

- This expression tests whether the current page URL contains the string `payment.html`:

```
$ ${system_integrations}.chat.browser_info.page_url.contains("payment.html")
```

You might use this expression in a step condition in order to avoid telling customers to navigate to a page they are already viewing. For example, in an action for paying a bill, you could have a step conditioned on this variable being `false` and has the following output: `First, click **Pay bill** to navigate to the bill payment page.`

`voice_telephony`

Included only if the phone integration is in use.

The `voice_telephony` object contains both response and request properties. Response properties are output values that are set by the phone integration and provide information about the call. Request properties are input values that you can modify in your actions to send logging data or to change configuration options for the call.

## Response properties (set by the phone integration)



**Note:** Properties contained in the `private` object are treated as private variables, which are not included in logs.

Name	Type	Description
<code>sip_call_id</code>	String	The SIP call ID associated with the phone call.
<code>assistant_phone_number</code>	String	The phone number associated with the Watson Assistant end of the call.
<code>sip_custom_invite_headers</code>	Object	A user-defined array of key/value pairs containing SIP headers from the SIP INVITE request.
<code>private.user_phone_number</code>	String	The phone number from which the customer's call originated.
<code>private.sip_request_uri</code>	String	The inbound SIP request URI that initiated the phone call.
<code>private.sip_from_uri</code>	String	The URI from the <code>From</code> header of the SIP request.
<code>private.sip_to_uri</code>	String	The URI from the <code>To</code> header of the SIP request.
<code>final_utterance_timeout_occurred</code>	Boolean	Set to <code>true</code> when the final utterance timeout has been reached. This timeout can be configured by sending the <code>final_utterance_timeout_count</code> property.
<code>post_response_timeout_occurred</code>	Boolean	Set to <code>true</code> when the final utterance timeout has been reached. This timeout can be configured by sending the <code>post_response_timeout_count</code> property.

### Response properties of the `voice_telephony` object

## Example response JSON

```

$ "voice_telephony": {
  "private": {
    "user_phone_number": "+18595553456",
    "sip_request_uri": "sips:+18885557777@public.voip.us-east.assistant.watson.cloud.ibm.com",
    "sip_from_uri": "sips:+18565558576@twilio.com",
    "sip_to_uri": "sips:+18885557777@public.voip.us-east.assistant.watson.cloud.ibm.com"
  },
  "sip_call_id": "Aob2-2743-5678-1234",
  "assistant_phone_number": "+18885556789",
  "sip_custom_invite_headers": {
    "custom-header1": "123",
    "custom-header2": "456"
  }
}

```

## Request properties (set by the assistant)

Name	Type	Description
final_utterance_timeout_count	Integer	The time (in milliseconds) to wait for a final utterance from the Speech to Text service. If no final utterance is received before the timeout occurs, the phone integration sends a message to the assistant with the <code>final_utterance_timeout_occurred</code> property set to <code>true</code> .
post_response_timeout_count	Integer	The time (in milliseconds) to wait for a new utterance after the last response is played. If no utterance is received before the timeout occurs, the phone integration sends a message to the assistant that includes the <code>post_response_timeout_occurred</code> property set to <code>true</code> .
cdr_custom_data	Object	A JSON object containing key/value pairs to be stored in the CDR record for the call. Each time this object is sent, its contents are merged with data sent previously during the call.
turn_settings.timeout_count	Integer	The time (in milliseconds) to wait for Watson Assistant to finish processing each conversation turn.

### Request properties of the `voice_telephony` object

## Example request JSON

```

"voice_telephony" : {
  "post_response_timeout_count":10000,
  "final_utterance_timeout_count":30000,
  "turn_settings": {
    "timeout_count": 5000
  },
  "cdr_custom_data" : {
    "custom_data_1": "data 1",
    "custom_data_2": "data 2"
  }
}

```

`text.messaging`

Included only if the SMS with Twilio integration is in use.

## Properties

 **Note:** Properties contained in the `private` object are treated as private variables, which are not included in logs.

Name	Type	Description
------	------	-------------

---

assistant\_phone\_number String The phone number associated with the Watson Assistant end of the conversation.

private.user\_phone\_number String The phone number from which the customer's SMS message originated.

#### Properties of the text.messaging object

## Example JSON

```
"text.messaging": {  
  "private": {  
    "user_phone_number": "+18595553456"  
  },  
  "assistant_phone_number": "+18885556789"  
}
```

whatsapp

Included only if the WhatsApp integration is in use.

## Properties

 **Note:** Properties contained in the `private` object are treated as private variables, which are not included in logs.

Name	Type	Description
assistant_phone_number	String	The phone number associated with the Watson Assistant end of the conversation.
private.user_phone_number	String	The phone number from which the customer's WhatsApp message originated.

#### Properties of the whatsapp object

## Example JSON

```
"whatsapp": {  
  "private": {  
    "user_phone_number": "+18595553456"  
  },  
  "assistant_phone_number": "+18885556789"  
}
```

slack

Included only if the Slack integration is in use.

## Properties

Name	Type	Description
team_id	String	The unique identifier of the Slack team.
channel_id	String	The unique identifier of the Slack channel.

## Example JSON

```
"slack": {  
  "team_id": "T02F3KE542J",  
  "channel_id": "C4K3KTTRD"  
}
```

facebook

Included only if the Facebook integration is in use.

## Properties

No additional properties.

## Example JSON

```
"facebook": {}
```

teams

Included only if the Microsoft Teams integration is in use.

## Properties

Name	Type	Description
conversation_id	String	The unique identifier of the Microsoft Teams conversation.

Properties of the teams object

## Example JSON

```
"teams": {  
  "conversation_id": "a:1ATy08jyGkPGy2QdKIrGZL5u_o6fIUVDRKeIZtkIUAkQDC23FC9S97f18i-UN1-  
  eISAfDWqoQeTbregvSE8jK0LNy6h9VssNcN3CsGG9guMiUB0EeSqxnnEFpAVzbkayR"  
}
```

# Expression language methods for actions

The Watson Assistant expression language can be used to specify values that are independent of, or derived from, values that are collected in steps or stored in session variables. You can use an expression to define a step condition or to define the value of a session variable.



**Note:** The Watson Assistant expression language is based on the Spring Expression Language (SpEL), but with some important differences in syntax. For detailed background information about SpEL, see [Spring Expression Language \(SpEL\)](#).

You can use SpEL expressions in two ways:

- Defining a step condition. For more information, see [Writing expressions](#).
- Assigning a value to a session variable. For more information, see [Using variables to manage conversation information](#).

## Referencing action variables

An action variable is created implicitly for any step that expects customer input, and the variable is bound to the step. To reference an action variable inside an expression, you must specify the step ID using the format  `${step_id}` (for example,  `${step_771}`). To find the step ID for a step, select the step and then look at the end of the URL in your browser.

## Referencing session variables

Session variables are created explicitly in the **Variables** section of the step. To reference a session variable inside an expression, you must specify the variable ID using the format  `${variable_id}` (for example,  `${current_time}`). You can find the variable ID in the list of variables. (For more information, see [Using variables to manage conversation information](#).)

When you are editing an expression, you can type `$` to see a drop-down list of variables you can reference. Select a variable from the list to automatically insert the step ID or variable ID.

## Supported data types

Expressions can use atomic JSON types (such as `integer`, `string`, `number`, and `boolean`), and compound data types (such as JSON arrays (`[]`) and objects (`{}`)). When specifying literal string values, you can use either single (`'`) or double (`"`) quotation marks.

Values that action steps collect from customers use customer response types such as date, time, currency, or percent. These values are stored as JSON objects in the following format:

```
{  
  "system_type": "{system_type}",  
  "value": "{value}"  
}
```

where `{system_type}` is one of the following types:

- `time`
- `percentage`
- `currency`

## Date and Time methods

Several methods are available to work with date and time values.

```
now(String timezone)
```

The `now()` method returns the current date and time for a specified time zone, in the format `yyyy-MM-dd HH:mm:ss 'GMT'XXX`:

```
$ now('Australia/Sydney').
```

In this example, if the current date and time is `2021-11-26 11:41:00`, the returned string is `2021-11-26 21:41:00 GMT+10.00` or `2021-11-26 21:41:00 GMT+11.00` depending on daylight saving time.

The output string format change above is applicable to date and time calculation methods as well. For example, if the `<date>` string used is in the format `yyyy-MM-dd HH:mm:ss`, such as when using the method `today()`, then the output is in the same format (`yyyy-MM-dd HH:mm:ss`). However, if the `<date>` string is in the format `yyyy-MM-dd HH:mm:ss 'GMT'XXX`, such when using the method `now()`, then the output will be in the format `yyyy-MM-dd HH:mm:ss 'GMT'XXX`.

```
.reformatDateTime(String format)
```

Formats date and time strings for output. The parameter is a format string specifying how the date or time value should be formatted. The format string must be specified using the Java [SimpleDateFormat](#) syntax.

This method returns a string formatted according to the specified format:

- `MM/dd/yyyy` for `12/31/2016`
- `h a` for `10pm`

To return the day of the week:

- `E` for `Tuesday`
- `u` for day index (`1 = Monday, ..., 7 = Sunday`)

For example, this expression returns the value `17:30:00` as `5:30 PM`:

```
 ${system_current_date}.reformatDateTime('h:mm a')
```

 **Note:** If the input string includes only a time, the default date `1970-01-01` is used in the output. If the input string includes only a date, the default time `12 AM (00:00)` is used.

```
.before(String date/time)
```

- Determines whether a date/time value is before the specified date/time argument, as in this example:

```
$ ${system_current_date}.before('2021-11-19')
```

You can compare a date with another date, or a time with another time. You can also compare a time with a date/time value, in which case the date is ignored and only the times are compared. Any other comparisons of mismatched values (for example, comparing a date with a time) return `false`, an exception is logged in `output.debug.log_messages` (which you can see in the assistant preview or in API responses).

```
.after(String date/time)
```

- Determines whether the date/time value is after the date/time argument.

```
.sameMoment(String date/time)
```

- Determines whether the date/time value is the same as the date/time argument.

```
.sameOrAfter(String date/time)
```

- Determines whether the date/time value is after or the same as the date/time argument.
- Analogous to `.after()`.

```
.sameOrBefore(String date/time)
```

- Determines whether the date/time value is before or the same as the date/time argument.

## Date and time calculations

Use the following methods to calculate a date.

Method	Description
<code>&lt;date&gt;.minusDays(_n_)</code>	Returns the date of the day <i>n</i> days before the specified date.
<code>&lt;date&gt;.minusMonths(_n_)</code>	Returns the date of the day <i>n</i> months before the specified date.
<code>&lt;date&gt;.minusYears(_n_)</code>	Returns the date of the day <i>n</i> years before the specified date.
<code>&lt;date&gt;.plusDays(_n_)</code>	Returns the date of the day <i>n</i> days after the specified date.
<code>&lt;date&gt;.plusMonths(_n_)</code>	Returns the date of the day <i>n</i> months after the specified date.
<code>&lt;date&gt;.plusYears(n)</code>	Returns the date of the day <i>n</i> years after the specified date.

where `<date>` is specified in the format `yyyy-MM-dd` or `yyyy-MM-dd HH:mm:ss`.

For example, to get tomorrow's date, specify the following expression:

```
 ${system_current_date}.plusDays(1)
```

Use the following methods to calculate a time.

Method	Description
<code>&lt;time&gt;.minusHours(_n_)</code>	Returns the time <i>n</i> hours before the specified time.
<code>&lt;time&gt;.minusMinutes(_n_)</code>	Returns the time <i>n</i> minutes before the specified time.
<code>&lt;time&gt;.minusSeconds(_n_)</code>	Returns the time <i>n</i> seconds before the specified time.
<code>&lt;time&gt;.plusHours(_n_)</code>	Returns the time <i>n</i> hours after the specified time.
<code>&lt;time&gt;.plusMinutes(_n_)</code>	Returns the time <i>n</i> minutes after the specified time.
<code>&lt;time&gt;.plusSeconds(_n_)</code>	Returns the time <i>n</i> seconds after the specified time.

where `<time>` is specified in the format `HH:mm:ss`.

For example, to get the time one hour from now, specify the following expression:

```
now().plusHours(1)
```

## Working with time spans

To show a response based on whether today's date falls within a certain time span, you can use a combination of time-related methods. For example, if you run a special offer during the holiday season every year, you might want to check whether today's date falls between November 25 and December 24 of this year.

First, define the dates of interest as session variables. In the following start and end date session variable expressions, the date is being constructed by concatenating the dynamically-derived current year value with hard-coded month and day values:

```
start_date = now().reformatDateTime('Y') + '-12-24'  
end_date = now().reformatDateTime('Y') + '-11-25'
```

Then, in a step condition, you can indicate that you want to show the response only if the current date falls between the start and end dates that you defined as session variables:

```
now().after(${start_date}) && now().before(${end_date})
```

### java.util.Date support

In addition to the built-in methods, you can use standard methods of the `java.util.Date` class.

For example, to get the date of the day that falls a week from today, you can use the following syntax:

```
new Date(new Date().getTime() + (7 * (24*60*60*1000L)))
```

This expression first gets the current date in milliseconds (since January 1, 1970, 00:00:00 GMT). It also calculates the number of milliseconds in 7 days (`(24*60*60*1000L)` represents one day in milliseconds). It then adds 7 days to the current date. The result is the full date of the day that falls a week from today (for example, `Fri Jan 26 16:30:37 UTC 2018`).

## Number methods

These methods help you get and reformat number values.

For information about recognizing numbers in customer responses, see [Choosing a response type](#).

If you want to change the decimal placement for a number (for example, to reformat a number as a currency value), see the [String format\(\) method](#).

### toDouble()

Converts the object or field to the Double number type. You can call this method on any object or field. If the conversion fails, `null` is returned.

### toInt()

Converts the object or field to the Integer number type. You can call this method on any object or field. If the conversion fails, `null` is returned.

### toLong()

Converts the object or field to the Long number type. You can call this method on any object or field. If the conversion fails `null` is returned.

To specify a Long number type in a SpEL expression, you must append an `L` to the number to identify it as such (for example, `5000000000L`). This syntax is required for any numbers that do not fit into the 32-bit Integer type. Numbers that are greater than  $2^{31}$  (2,147,483,648) or lower than  $-2^{31}$  (-2,147,483,648) are considered Long number types. Long number types have a minimum value of  $-2^{63}$  and a maximum value of  $2^{63}-1$  (or 9,223,372,036,854,775,807).

## Standard math

Use SpEL expressions to define standard math equations, where the operators are represented by using these symbols:

Arithmetic operation	Symbol
addition	<code>+</code>
division	<code>/</code>
multiplication	<code>*</code>
subtraction	<code>-</code>

`java.lang.Math()`

You can use the functions of the `java.lang.Math` class to perform basic numeric operations.

You can use the the Class methods, including these:

- `max()`:

`T(Math).max(${step_297},${step_569})`

- `min()`:

`T(Math).min(${step_297},${step_569})`

- `pow()`:

`T(Math).pow(${step_297}.toDouble(),2.toDouble())`

See the [java.lang.Math reference documentation](#) for information about other methods.

`java.util.Random()`

Returns a random number. You can use any of the following syntax options:

- To return a random boolean value (`true` or `false`), use `new Random().nextBoolean()`.
- To return a random double number between 0 (included) and 1 (excluded), use `new Random().nextDouble()`.
- To return a random integer between 0 (included) and a number you specify, use `new Random().nextInt(_n_)`, where `n` is 1 greater than the top of the number range you want. For example, if you want to return a random number between 0 and 10, specify `new Random().nextInt(11)`.
- To return a random integer from the full integer value range (-2147483648 to 2147483648), use `new Random().nextInt()`.

For example, you might create step that is executed only for a randomly selected subset of customers. The following step condition would mean that the step has a 50% chance of executing:

```
new Random().nextInt(2) == 1
```

See the [java.util.Random reference documentation](#) for information about other methods.

You can also use standard methods of the following classes:

- `java.lang.Byte`
- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Double`
- `java.lang.Short`
- `java.lang.Float`

## String methods

---

These methods help you work with text.



**Note:** For details about the syntax to use in methods that involve regular expressions, see [RE2 Syntax reference](#).

`String.append(Object)`

This method appends an input object (as a string) to a string, and returns a modified string.

```
 ${step_297}.append('next text')
```

`String.contains(String)`

This method returns `true` if the action variable or session variable contains a substring. This is useful in conditions.

```
 ${step_297}.contains('Yes')
```

`String.endsWith(String)`

This method returns `true` if the string ends with the input substring.

```
 ${step_297}.endsWith('?')
```

`String.equals(String)`

This method returns `true` if the specified string equals the action variable or session variable.

```
 ${step_297}.equals('Yes')
```

`String.equalsIgnoreCase(String)`

This method returns `true` if the specified string equals the action variable or session variable irrespective of case.

```
 ${step_297}.equalsIgnoreCase('Yes')
```

`String.extract(String regexp, Integer groupIndex)`

This method returns a string from the input that matches the specified regular expression group pattern. It returns an empty string if no match is found.



**Note:** This method is designed to extract matches for different regex pattern groups, not different matches for a single regex pattern. To find different matches, see the [getMatch\(\)](#) method.

In this example, the action variable is saving a string that matches the regex pattern group that you specify. In the expression, two regex patterns groups are defined, each one enclosed in parentheses. There is an inherent third group that is comprised of the two groups together. This is the first (groupIndex 0) regex group; it matches with a string that contains the full number group and text group together. The second regex group (groupIndex 1) matches with the first occurrence of a number group. The third group (groupIndex 2) matches with the first occurrence of a text group after a number group.

```
 ${step_297}.extract('([\d]+)([A-Za-z]+)', <n>)
```

If action varibale contains:

```
Hello 123 this is 456.
```

the results are as follows:

- When <n>=0, the value is 123 this.
- When <n>=1, the value is 123.
- When <n>=2, the value is this.

```
String.find(String regexp)
```

This method returns `true` if any segment of the string matches the input regular expression. You can call this method against a `JSONArray` or `JSONObject` element, and it will convert the array or object to a string before making the comparison.

For example, if the action variable  `${step_297}` collects the string `Hello 123456`, then the the following expression returns `true`:

```
 ${step_297}.find('^[^\d]*[\d]{6}[^$\d]*$')
```

The condition is `true` because the numeric portion of the input text matches the regular expression `^[^\d]*[\d]{6}[^$\d]*$`.

```
String.getMatch(String regexp, Integer matchIndex)
```

This method returns a string that matches the occurrence of the specified regular expression pattern. This method returns an empty string if no match is found.

As matches are found, they are added to what you can think of as an array of matches. Because the array element count starts at 0, if you want to return the third match, you would specify 2 as the `matchIndex` value. For example, if you enter a text string with three words that match the specified pattern, you can return the first, second, or third match only by specifying its index value.

For example, the following example is looking for a group of numbers in an action variable.

```
 ${step_297}.getMatch('([\d]+)', 1)
```

If the action variable  `${step_297}` contains the string `hello 123 i said 456 and 8910`, this expression returns `456`.

```
String.isEmpty()
```

This method returns `true` if the string is an empty string, but not `null`, as in this example:

```
 ${step_297}.isEmpty()
```

```
String.length()
```

This method returns the character length of the string, as in this example:

```
 ${step_297}.length()
```

If the action variable  `${step_297}` contains the string `Hello`, this expression returns 5.

```
String.matches(String regexp)
```

This method returns `true` if the string matches the input regular expression, as in the example:

```
 ${step_297}.matches('^Hello$')
```

If the action variable  `${step_297}` contains the string `Hello`, this expression evaluates to `true`.

```
String.startsWith(String)
```

This method returns `true` if the string starts with the specified substring, as in this example:

```
 ${step_297}.startsWith('What')
```

If the action variable  `${step_297}` contains the string `What is your name?`, this expression returns `true`.

```
String.substring(Integer beginIndex, Integer endIndex)
```

This method returns a substring beginning with the character at `beginIndex` and ending with the character before `endIndex`. (The `endIndex` character itself is not included in the substring.) Note that the index values are zero-based, so the first character in the string is at index 0.

This example returns a substring that begins at index 5 (which is the sixth character), and continuing to the end of the string:

```
 ${step_297}.substring(5, ${step_297}.length())
```

If the action variable  `${step_297}` contains the string `This is a string.`, this expression returns `is a string.`

```
String.toJson()
```

This method parses a string that contains JSON data and returns a JSON object or array, as in this example:

```
 $ ${json_var}.toJson()
```

If the session variable  `${json_var}` contains the following string:

```
 $ "{\"firstname\": \"John\", \"lastname\": \"Doe\" }"
```

the `toJson()` method returns the following object:

```
 $ {  
  "firstname": "John",  
  "lastname": "Doe"  
}
```

```
String.toLowerCase()
```

This method returns the specified string converted to lowercase letters, as in this example:

```
`${step_297}.toLowerCase()
```

If the action variable ``${step_297}` contains the string `This is A DOG!`, this expression returns the string `this is a dog!`.

```
String.toUpperCase()
```

This method returns the original string converted to uppercase letters, as in this example:

```
`${step_297}.toUpperCase()
```

If the action variable ``${step_297}` contains the string `hi there`, this method returns the string `HI THERE`.

```
String.trim()
```

This method trims any spaces at the beginning and end of a string and returns the modified string, as in this example:

```
`${step_297}.trim()
```

If the action variable ``${step_297}` contains the string `something is here`, this method returns the string `something is here`.

## java.lang.String **support**

In addition to the built-in methods, you can use standard methods of the `java.lang.String` class.

```
java.lang.String.format()
```

You can apply the standard Java String `format()` method to text. For information about the syntax to use, see [Format String Syntax](#).

This example takes three decimal integers (1, 1, and 2) and adds them to a sentence:

```
T(java.lang.String).format('%d + %d equals %d', 1, 1, 2)
```

The resulting string is `1 + 1 equals 2`.

This example changes the decimal placement for a number collected by a step:

```
T(String).format('%.2f', ${step_297})
```

If the ``${step_297}` variable that needs to be formatted in US dollars is 4.5, the resulting string is `4.50`.

## Array methods

These methods help you work with arrays.

```
Array.add(value...)
```

This method adds one or more new values to the array and returns `true` if the operation is successful.

```
`${Items}.add('four', 'five')
```

If `Items` is `['one', 'two', 'three']`, this example updates it in place to become `['one', 'two', 'three', 'four', 'five']`.

```
Array.addAll(Array array)
```

This method adds one array to another and returns `null`.

```
 ${Items}.addAll(${New_items})
```

If `Items` is `['one', 'two', 'three']` and `New_items` is `['four', 'five', 'six']`, this example updates `Items` in place to become `['one', 'two', 'three', 'four', 'five', 'six']`.

```
Array.append(value...)
```

This method appends one or more new values to the array and returns the result as a new array. The original array is not modified.

```
 ${Items}.append('four', 'five')
```

If `Items` is `['one', 'two', 'three']`, this example returns the new array `['one', 'two', 'three', 'four', 'five']`.

```
Array.clear()
```

This method removes all values from the array and returns `null`.

```
 $ ${Items}.clear()
```

After this expression is evaluated, `Items` is an empty array (`[]`).

```
Array.contains(value)
```

This method returns `true` if the array contains an item that is exactly equal to the input value. The specified value can be a string or a number.

```
 $ ${Items}.contains(123)
```

If `Items` is `[123, 456, 789]`, this example returns `true`.

```
Array.containsIgnoreCase(value)
```

This method returns `true` if the array contains an item that is equal to the input value. Strings are matched regardless of whether the value is specified in uppercase or lowercase letters. The specified value can be a string or a number.

```
 $ ${Items}.contains('two')
```

This example returns `true` if the `Items` array contains any capitalization of the string `two` (for example, `TWO` or `Two` would also match).

```
Array.filter(temp_var, "temp_var.property operator comparison_value")
```

Filters an array by comparing each array element to a value you specify, returning a new array that contains only the matching elements.

The filter expression consists of the following values:

- `temp_var`: An arbitrary name for a temporary variable that is used to hold each array element as it is evaluated. For example, if the original array contains objects describing cities, you might use `city` as the temporary variable name.
- `property`: The element property that you want to filter on. This must be a property of the elements in the source array. Specify the property as a property of `temp_var`, using the syntax `temp_var.property`. For example, if `latitude` is a

valid property name for the source elements, you might specify the property as `city.latitude`.

- `operator`: The operator to use to compare the property value to the comparison value. You can use any of the following operators:

Operator	Description
<code>==</code>	Is equal to
<code>&gt;</code>	Is greater than
<code>&lt;</code>	Is less than
<code>&gt;=</code>	Is greater than or equal to
<code>&lt;=</code>	Is less than or equal to
<code>!=</code>	Is not equal to

#### Supported filter operators

- `comparison_value`: The value that you want to compare each array element property value to. You can specify a literal value or reference a variable.

## Filter examples

For example, you might have an array of objects containing city names and their population numbers:

```
[  
  {  
    "name": "Tokyo",  
    "population": 13988129  
  },  
  {  
    "name": "Rome",  
    "population": 2860009  
  },  
  {  
    "name": "Beijing",  
    "population": 21893095  
  },  
  {  
    "name": "Paris",  
    "population": 2165423  
  }]
```

If the source array is stored in a variable called  `${cities}`, the following expression returns a smaller array that contains only cities with populations greater than 5 million:

```
 ${cities}.filter("city", "city.population > 5000000")
```

The expression returns the following filtered array:

```
[  
  {  
    "name": "Tokyo",  
    "population": 13988129  
  },  
  {  
    "name": "Beijing",  
    "population": 21893095  
  }  
]
```

Instead of a hardcoded comparison value, you could also filter based on a dynamic value stored in a variable. This example filters using a population value specified by a customer response in a previous step:

```
 ${cities}.filter("city", "city.population > ${step_123}")
```

**Tip:** When comparing number values, be sure to set the context variable involved in the comparison to a valid value before the filter method is triggered. Note that `null` can be a valid value if the array element you are comparing it against might contain it.

`Array.get(Integer index)`

This method returns the item from the array that is at the specified index position. Note that arrays are zero-indexed, meaning that the first item in the array is at index position `0`.

```
 $ ${Items}.get(1)
```

If `Items` is `['one', 'two', 'three']`, this example returns `two`.

The `get()` method is an alternative to using brackets (`[]`) to retrieve an item from an array. The following example is also valid and returns the same result:

```
 $ ${Items}[1]
```

**Note:** If you are using a value specified by a customer to choose an item from an array, you might need to subtract `1` to convert to a zero-indexed value. For example, you might use an expression like  `${Items}.get(${step_123} - 1)` to retrieve the intended value.

`Array.getRandomItem()`

This method returns a randomly chosen item from the array.

```
 $ ${Items}.getRandomItem()
```

If `Items` is `['one', 'two', 'three']`, this example returns `one`, `two`, or `three`, on a random basis.

`Array.indexOf(value)`

This method returns the index position of the first occurrence of the input value in the array, or `-1` if the array does not contain the input value. The specified value can be a string or a number.

```
 $ ${Items}.indexOf('two')
```

If `Items` is `['one', 'two', 'three']`, this example returns the integer `1` (indicating the second position in the zero-indexed array).

```
Array.join(String delimiter)
```

This method joins all values in this array to a string. Values are converted to string and delimited by the input delimiter.

For example, you might have a variable called `pizza_toppings` that contains the array `["pepperoni", "ham", "mushrooms"]`. The following expression converts this array into the string `pepperoni, ham, mushrooms`:

```
$ ${toppings_array}.join(', ')
```

If you use that expression to define the value of a variable, you can then reference that variable in your assistant output to create a human-readable message (for example, `You have selected the following toppings: pepperoni, ham, mushrooms`).

```
JSONArray.joinToArray(template, retainDataType)
```

This method extracts information from each item in the array and builds a new array that is formatted according to the template you specify. The template can be a string, a JSON object, or an array. The method returns an array of strings, an array of objects, or an array of arrays, depending on the type of template.

This method is useful for formatting information as a string you can return as part of the output of a step, or for transforming data into a different structure so you can use it with an external API.

In the template, you can reference values from the source array using the following syntax:

```
$ %e.{property}%
```

where `{property}` represents the name of the property in the source array.

For example, suppose your assistant has stored an array containing flight details in a session variable. The stored data might look like this:

```
"flights": [
  {
    "flight": "AZ1040",
    "origin": "JFK",
    "carrier": "Alitalia",
    "duration": 485,
    "destination": "FCO",
    "arrival_date": "2019-02-03",
    "arrival_time": "07:00",
    "departure_date": "2019-02-02",
    "departure_time": "16:45"
  },
  {
    "flight": "DL1710",
    "origin": "JFK",
    "carrier": "Delta",
    "duration": 379,
    "destination": "LAX",
    "arrival_date": "2019-02-02",
    "arrival_time": "10:19",
    "departure_date": "2019-02-02",
    "departure_time": "07:00"
  },
  {
    "flight": "VS4379",
    "origin": "BOS",
    "carrier": "Virgin Atlantic",
    "duration": 385,
```

```
        "destination": "LHR",
        "arrival_date": "2019-02-03",
        "arrival_time": "09:05",
        "departure_date": "2019-02-02",
        "departure_time": "21:40"
    }
]
```

To build an array of strings that describe these flights in a user-readable form, you might use the following expression:

```
$ ${Flight_data}.joinToArray("Flight %e.flight% to %e.destination%", true)
```

This expression would return the following array of strings: ["Flight AZ1040 to FCO", "Flight DL1710 to LAX", "Flight VS4379 to LHR"].

The optional `retainDataType` parameter specifies whether the method should preserve the data type of all input values in the returned array. If `retainDataType` is set to `false` or omitted, in some situations, strings in the input array might be converted to numbers in the returned array. For example, if the selected values from the input array are `"1"`, `"2"`, and `"3"`, the returned array might be `[ 1, 2, 3 ]`. To avoid unexpected type conversions, specify `true` for this parameter.

## Complex templates

A more complex template might contain formatting that displays the information in a legible layout. For a complex template, you might want to store the template in a session variable, which you can then pass to the `joinToArray` method instead of a string.

For example, this complex template contains a subset of the array elements, adding labels and formatting:

```
<br/>Flight number: %e.flight% <br/> Airline: %e.carrier% <br/> Departure date: %e.departure_date%
<br/> Departure time: %e.departure_time% <br/> Arrival time: %e.arrival_time% <br/>
```



**Note:** Make sure any formatting you use in your template is supported by the channel integration that will be displaying the assistant output.

If you create a session variable called `Template`, and assign this template as its value, you can then use that variable in your expressions:

```
$ ${Flight_data}.joinToArray(${Template})
```

At run time, the response would look like this:

```
Flight number: AZ1040
Airline: Alitalia
Departure date: 2019-02-02
Departure time: 16:45
Arrival time: 07:00

Flight number: DL1710
Airline: Delta
Departure date: 2019-02-02
Departure time: 07:00
Arrival time: 10:19

Flight number: VS4379
Airline: Virgin Atlantic
Departure date: 2019-02-02
Departure time: 21:40
Arrival time: 09:05
```

## JSON templates

Instead of a string, you can define a template as a JSON object. This provides a way to standardize the formatting of information from different systems, or to transform data into the format required for an external service.

In this example, a template is defined as a JSON object that extracts flight details from the elements specified in the array stored in the `Flight data` session variable:

```
{  
  "departure": "Flight %e.flight% departs on %e.departure_date% at %e.departure_time%.",  
  "arrival": "Flight %e.flight% arrives on %e.arrival_date% at %e.arrival_time%."  
}
```

Using this template, the `joinToArray()` method returns a new array of objects with the specified structure.

```
Array.remove(Integer index)
```

This method removes the item in the specified index position from the array, and returns the updated array.

```
$ ${Items}.remove(1)
```

If `Items` is `['one', 'two', 'three']`, this example returns `['one', 'three']`. The original `Items` array is also modified in place.

```
Array.removeValue(value)
```

This method removes the first occurrence of the specified value from the array, and returns the updated array. The specified value can be a string or a number.

```
$ ${Items}.removeValue('two')
```

If `Items` is `['one', 'two', 'three']`, this example returns `['one', 'three']`. The original `Items` array is also modified in place.

```
Array.set(Integer index, value)
```

This method replaces the item in the specified index position with the specified value, and returns the updated array.

```
$ ${Items}.set(2,'five')
```

If `Items` is `['one', 'two', 'three']`, this example returns `['one', 'two', 'five']`. The original `Items` array is also modified in place.

```
Array.size()
```

This method returns the number of items in the array as an integer.

```
$ ${Items}.size()
```

If `Items` is `['one', 'two', 'three']`, this example returns `3`.

# Phone integration context variables

You can use context variables to manage the flow of conversations with customers who interact with your assistant over the telephone.

The following tables describe the context variables that have special meaning in the context of the phone integration. They should not be used for any purpose other than the documented use.

## Context variables that are set by the phone channel

Name	Type	Description
sip_call_id	string	The SIP call ID associated with the Watson Assistant session.
sip_custom_invite_headers	object	A JSON object containing key/value pairs defining SIP headers that are pulled from the initial SIP INVITE request and passed to the Watson Assistant service (for example, <code>{"Custom-Header1": "123"}</code> ).
private.sip_from_uri	string	The SIP <code>From</code> URI associated with the Watson Assistant service.
private.sip_request_uri	string	The SIP request URI that started the conversation session.
private.sip_to_uri	string	The SIP <code>To</code> URI associated with the conversation session.
private.user_phone_number	string	The phone number that the call was received from.
assistant_phone_number	string	The phone number associated with the Watson Assistant side that received the phone call.

Context variables set by the phone channel

## Input parameters that are set by the phone channel

The following input parameters are only valid for the current conversation turn.

Name	Type	Description
post_response_timeout_occurred	boolean	Whether the post response timeout expired.
barge_in_occurred	boolean	Whether barge-in occurred.
final_utterance_timeout_occurred	true or false	Whether the final utterance timeout expired.
dtmf_collection_succeeded	boolean	Whether the DTMF collection succeeded or failed. When <code>true</code> , a DTMF collection succeeded, and returns the expected number of digits. When <code>false</code> , a DTMF collection failed to collect the specified number of digits. Even when <code>dtmf_collection_succeeded</code> is <code>false</code> , all collected digits are passed to the dialog in the input string of the turn request.
is_dtmf	boolean	Whether the input to Watson Assistant is dual-tone multi-frequency signaling (DTMF).

speech_to_text_result	object	The final response from the Speech to Text service in JSON format, including the transcript and confidence score for the top hypothesis and any alternatives. The format matches exactly the format that is received from the Speech to Text service. (For more information, see the <a href="#">Speech to Text API documentation</a> .)
-----------------------	--------	--

#### Input parameters set by the phone channel

## Example

```
{
  "input": {
    "text": "agent",
    "integrations": {
      "voice_telephony": {
        "speech_to_text_result": {
          "result_index": 0,
          "stopTimestamp": "2021-09-29T17:43:31.036Z",
          "transaction_ids": {
            "x-global-transaction-id": "43dd6ce0-139a-4d76-95aa-86e03fcfc434",
            "x-dp-watson-tran-id": "6e60695e-fed7-4efe-a376-0888b027d30f"
          },
          "results": [
            {
              "final": true,
              "alternatives": [
                {
                  "transcript": "agent",
                  "confidence": 0.78
                }
              ]
            }
          ],
          "transactionID": "43dd6ce0-139a-4d76-95aa-86e03fcfc434",
          "startTimestamp": "2021-09-29T17:43:29.436Z"
        },
        "is_dtmf": false,
        "barge_in_occurred": false
      }
    }
  },
  "context": {
    "skills": {
      "main_skill": {
        "user_defined": {},
        "system": {}
      }
    },
    "integrations": {
      "voice_telephony": {
        "private": {
          "sip_to_uri": "sip:watson-conversation@10.10.10.10",
          "sip_from_uri": "sip:10.10.10.11",
          "sip_request_uri": "sip:test@10.10.10.10:5064;transport=tcp"
        },
        "sip_call_id": "QjryZsuAS4",
        "assistant_phone_number": "18882346789"
      }
    }
  }
}
```

# SMS integration reference

Add action commands to the message `context` object to manage the flow of conversations with customers who interact with your assistant by submitting SMS messages over the telephone.

Learn about the supported commands and reserved context variables that are used by the SMS integration.

## Supported commands

Each action consists of a `command` property, followed by an optional `parameter` property to define parameters for commands that require them. The commands that are described in the following table are supported by the *SMS* integration.

Action command	Description	Parameters
<code>terminateSession</code>	Ends the current SMS session. Use this command to ensure that the subsequent text message starts a new assistant-level session which does not retain any context values from the current session.	None
<code>smsActSendMedia</code>	Enables MMS messaging.	<code>mediaURL</code> : Specifies a JSON array of publicly accessible media URLs that are sent to the user.
<code>smsActSetDisambiguationConfig</code>	Configures how to handle the choices that are displayed in a disambiguation list.	<code>prefixText</code> : Text to include before each option. For example, <code>Press %s for</code> where <code>%s</code> represents the number corresponding to a list choice; this is replaced with the actual number at run time.
<code>smsActSetOptionsConfig</code>	Configures how to handle option response types.	<code>prefixText</code> : Text to include before each option. For example, <code>Press %s for</code> where <code>%s</code> represents the number corresponding to a list choice; this is replaced with the actual number at run time.

Table 1. Actions that you can initiate from the action

## Reserved context variables

The following table describes the context variables that have special meaning in the context of the SMS integration. They should not be used for any purpose other than the documented use.

Table 2 describes the context variables that are set by your action. Table 3 describes the context variables that you can set by the SMS integration.

## Table 2. Context variables that are set by your action

Context variable name	Expected value	Description
<code>smsConversationResponseTimeout</code>	Time in ms	The amount of time in milliseconds that the integration waits to receive a response from the action. If the time limit is exceeded, the integration attempts to contact the action again. If the service still can't be reached, the SMS response fails.

Table 2. SMS context variables set by the action

**Table 3. Context variables that are set by the integration**

Context variable name	Description
<code>smsTenantPhoneNumber</code>	The integration tenant phone number that the user is messaging.
<code>smsUserPhoneNumber</code>	The phone number of the user that is exchanging messages with the integration.
<code>smsUserData</code>	Data in JSON format to be passed verbatim to the service orchestration engine or Watson Assistant service. This variable is sent only if the session is started from the integration tenant and the data is sent through the REST API.
<code>smsSessionTimeoutCount</code>	The session timeout value. This variable is sent only if the timeout value is defined through the REST API.
<code>smsError</code>	When the integration fails to send an SMS message, this variable contains details about the error that occurred.
<code>smsSessionID</code>	The globally unique identifier (GUID) for the related SMS Gateway session.
<code>smsMedia</code>	The <code>arraylist</code> of <code>mediaURL</code> and corresponding <code>mediaContentType</code> . This context variable is cleared at the end of each conversation turn.

Table 3. SMS context variables set by the integration

# CDR log event reference

For `cdr_logged` events sent by a log webhook, the `payload` object contains data about a Call Detail Record (CDR) event that was handled by the phone integration. The `payload` object for a call detail record event contains the following properties:

Property	Type	Description
<code>primary_phone_number</code>	string	The phone number that was called.
<code>global_session_id</code>	string	The unique session identifier.
<code>failure_occurred</code>	boolean	Whether a failure occurred during the call.
<code>failure_details</code>	string	Details about any failure that occurred.
<code>transfer_occurred</code>	boolean	Whether an attempt was made to transfer a call.
<code>active_calls</code>	Number	The number of active calls when the call started.
<code>x-global-sip-trunk-call-id</code>	string	The value of the SIP trunk call ID header extracted from the initial SIP <code>INVITE</code> request. The following SIP trunk call ID headers are supported: <ul style="list-style-type: none"><li>• <code>X-Twilio-CallSid</code></li><li>• <code>X-SID</code></li><li>• <code>X-Global-SIP-Trunk-Call-ID</code></li></ul>
<code>call</code>	Object	Information about the call. See <a href="#">call</a> .
<code>session_initiation_protocol</code>	Object	SIP protocol related details. See <a href="#">session_initiation_protocol</a> .
<code>max_response_milliseconds</code>	Object	Maximum latency for services used during the call. See <a href="#">max_response_milliseconds</a> .
<code>assistant_interaction_summaries</code>	Array	Details about the Watson Assistant interactions that took place during the call. See <a href="#">assistant_interaction_summaries</a> .
<code>injected_custom_data</code>	Object	A set of key/value pairs extracted from the <code>cdr_custom_data</code> context variable.
<code>warnings_and_errors</code>	Array	Any warnings or errors that were logged during the call. See <a href="#">warnings_and_errors</a> .
<code>realtime_transport_network_summary</code>	Object	Statistics for the inbound stream in the <code>inbound_stream</code> object and statistics for the outbound stream in the <code>outbound_stream</code> object. Included only if RTCP is enabled. See <a href="#">realtime_transport_network_summary</a> .

## Properties of the CDR webhook payload object

`call`

The `call` object contains the following properties:

Property	Type	Description
<code>start_timestamp</code>	String	The time when the call started, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
<code>stop_timestamp</code>	String	The time when the call ended, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
<code>milliseconds_elapsed</code>	Number	The duration of the call, in milliseconds.
<code>end_reason</code>	string	The reason the call ended. The possible reasons are as follows: <ul style="list-style-type: none"><li>assistant_transfer</li><li>assistant_hangup</li><li>caller_hangup</li><li>failed</li></ul>
<code>security.media_encrypted</code>	Boolean	Whether the media was encrypted.
<code>security.signaling_encrypted</code>	Boolean	Whether the SIP signaling was encrypted.
<code>security.sip_authenticated</code>	Boolean	Whether SIP authentication was used to authenticate the caller.

#### Properties of the call object

`session_initiation_protocol`

The `session_initiation_protocol` object contains the following properties:

Property	Type	Description
<code>invite_arrival_timestamp</code>	String	The time when the <code>INVITE</code> request arrived, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
<code>setup_milliseconds</code>	Number	The time it took to set up the call, in milliseconds. This is the time between when the initial SIP <code>INVITE</code> request was received and when the final SIP <code>ACK</code> request was received.
<code>headers.call_id</code>	String	The SIP <code>Call-ID</code> header field pulled from the SIP <code>INVITE</code> related to the call.
<code>headers.from_uri</code>	String	The SIP URI from the initial SIP <code>INVITE From</code> header.
<code>headers.to_uri</code>	String	The SIP URI from the initial SIP <code>INVITE To</code> header.

#### Properties of the session\_initiation\_protocol object

`assistant_interaction_summaries`

The `assistant_interaction_summaries` object contains the following properties:

Property	Type	Description

---

assistant_id	String	The unique identifier of the assistant.
session_id	String	The unique identifier of the session.
turns	Array	An array of objects describing the Watson Assistant interactions that took place during the conversation. See <a href="#">assistant_interaction_summaries.turns[]</a> .

---

#### Properties of the assistant\_interaction\_summaries object

`assistant_interaction_summaries.turns[]`

The objects in the `assistant_interaction_summaries.turns` array contain the following properties:

Property	Type	Description
assistant.log_id	String	A unique identifier for the logged event. This can be used to correlate between message logs and CDR events.
assistant.start_timestamp	String	The time when the request was sent to the assistant, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
assistant.response_milliseconds	Number	The time (in milliseconds) between when the request was sent and when the response was received from the assistant.
request	Object	A request sent to the assistant. See <a href="#">assistant_interaction_summaries.turns[].request</a> .
response	Array	An array of the response objects associated with the request.

---

#### Properties of the objects in the `assistant_interaction_summaries.turns[]` array

`assistant_interaction_summaries.turns[].request`

The `assistant_interaction_summaries.turns[] .request` object contains the following properties:

Property	Type	Description
----------	------	-------------

---

type	String	The request type: <ul style="list-style-type: none"><li>start: an initial request is sent to the assistant</li><li>speech_to_text: input is received from the Speech to Text service</li><li>dtmf: DTMF collection completes</li><li>sms: an SMS message is received from the caller</li><li>post_response_timeout: the post-response timer expires</li><li>redirect: a call is redirected</li><li>transfer: a call is transferred</li><li>transfer_failed: a call transfer fails</li><li>final_utterance_timeout: the final utterance timer expires</li><li>no_input_turn: no input turn is enabled</li><li>sms_failure: an SMS message cannot be sent to the caller</li><li>speech_to_text_result_filtered: an utterance is filtered due to low confidence level</li><li>mrcp_recognition_unsuccessful: the MRCP recognition completes without a final utterance</li><li>network_warning: a network error is detected</li><li>media_capability_change: media capabilities change during a call</li></ul>
------	--------	--

streaming_statistics	Object	Information and statistics related to the Speech to Text recognition. See <a href="#">assistant_interaction_summaries.turns[].request.streaming_statistics</a> .
----------------------	--------	--

#### Properties of the `assistant_interaction_summaries.turns[].request` object

`assistant_interaction_summaries.turns[].request.streaming_statistics`

The `assistant_interaction_summaries.turns[].request.streaming_statistics` object contains the following properties:

Property	Type	Description
<code>transaction_id</code>	String	A unique identifier of the transaction.
<code>start_timestamp</code>	String	The time when the transaction started, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
<code>stop_timestamp</code>	String	The time when the transaction ended, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
<code>response_milliseconds</code>	Number	The latency (in milliseconds) between when silence is detected in the caller's speech and a final result from the assistant is received.
<code>echo_detected</code>	Boolean	Whether an echo was detected.
<code>confidence</code>	Number	The confidence score of the final utterance.

#### Properties of the `assistant_interaction_summaries.turns[].streaming_statistics` object

`assistant_interaction_summaries.turns[].response`

The `assistant_interaction_summaries.turns[].response` object contains the following properties:

Property	Type	Description
----------	------	-------------

---

<code>type</code>	String	The response type: <ul style="list-style-type: none"><li><code>text_to_speech</code>: a command to play an utterance to the caller</li><li><code>sms</code>: a command to send an SMS message to the caller</li><li><code>url</code>: a command to play an audio file to the caller</li><li><code>transfer</code>: a command to transfer a call</li><li><code>text_to_speech_config</code>: a command to change the Text to Speech settings</li><li><code>speech_to_text_config</code>: a command to change the Speech to Text settings</li><li><code>pause_speech_to_text</code>: a command to stop speech recognition</li><li><code>unpause_speech_to_text</code>: a command to start speech recognition</li><li><code>pause_dtmf</code>: a command to stop processing of inbound DTMF signals</li><li><code>unpause_dtmf</code>: <code>unpause_dtmf</code>: a command to start processing of inbound DTMF signals</li><li><code>enable_speech_barge_in</code>: a command to enable speech barge-in so that callers can interrupt playback by speaking</li><li><code>disable_speech_barge_in</code>: a command to disable speech barge-in so that playback isn't interrupted when callers speak during audio playback</li><li><code>enable_dtmf_barge_in</code>: a command to enable DTMF barge-in so that callers can interrupt playback from the phone integration by pressing a key</li><li><code>disable_dtmf_barge_in</code>: a command to disable DTMF barge-in so that playback from the phone integration isn't interrupted when the caller presses a key</li><li><code>dtmf</code>: a command to send DTMF signals to the caller</li><li><code>hangup</code>: a command to disconnect the call</li></ul> See <a href="#">Mapping between CDR and Watson Assistant response types</a> .
<code>barge_in_occurred</code>	Boolean	Whether barge-in occurred during the turn.
<code>streaming_statistics</code>	Object	Information and statistics related to Text to Speech synthesis and playback. See <a href="#">assistant_interaction_summaries.turns[].response.streaming_statistics</a> .

Properties of the `assistant_interaction_summaries.turns[].response` object

## Mapping between CDR and Watson Assistant response types

The values of the `type` property map to Watson Assistant response types as follows:

CDR response type	Watson Assistant response type
<code>text_to_speech</code>	<code>text</code>
<code>url</code>	<code>audio</code>
<code>dtmf</code>	<code>dtmf, command_info.type:send</code>
<code>sms</code>	<code>user_defined, vgwAction.command:vgwActSendSMS</code>
<code>transfer</code>	<code>connect_to_agent</code>
<code>text_to_speech_config</code>	<code>text_to_speech, command_info.type:configure</code>
<code>speech_to_text_config</code>	<code>speech_to_text, command_info.type:configure</code>
<code>unpause_speech_to_text</code>	<code>start_activities, type:speech_to_text_recognition</code>

pause_speech_to_text	stop_activities, type:speech_to_text_recognition
unpause_dtmf	start_activities, type:dtmf_collection
pause_dtmf	stop_activities, type:dtmf_collection
enable_speech_barge_in	text_to_speech, command_info.type:enable_barge_in
disable_speech_barge_in	text_to_speech, command_info.type:disable_barge_in
enable_dtmf_barge_in	dtmf, command_info.type:enable_barge_in
disable_dtmf_barge_in	dtmf, command_info.type:disable_barge_in
hangup	end_session

assistant\_interaction\_summaries.turns[].response.streaming\_statistics

The `assistant_interaction_summaries.turns[].response.streaming_statistics` object contains the following properties:

Property	Type	Description
transaction_id	String	A unique identifier of the transaction.
start_timestamp	String	The time when the transaction started, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
stop_timestamp	String	The time when the transaction ended, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
response_milliseconds	Number	The time (in milliseconds) between when a text utterance is sent to the assistant and when the phone integration receives the first packet of synthesized audio.

#### Properties of the `assistant_interaction_summaries.turns[].response.streaming_statistics` object

`warnings_and_errors`

The `warnings_and_errors` object contains warnings and errors that were logged during the call, listed in order of occurrence. Warnings for the following conditions are included:

- Messages when utterances are filtered out by the confidence score threshold.
- Text to Speech underflows, which is when Text to Speech synthesis can't keep up with the phone integration streaming rate and audio might skip.
- RTP network warnings, such as high packet loss or high average jitter, if RTCP is enabled.

The following example shows the structure of the `warnings_and_errors` object:

```
$  "warnings_and_errors": [
  {
    "message": "CWSMR0032W: A Watson Speech to Text final utterance has a confidence score of 0.1, which does not meet the confidence score threshold of 0.2. The utterance will be ignored.",
    "id": "CWSMR0032W"
  },
  ...
]
```

```
{
  "message": "CWSMR0031W: The synthesis stream from the Watson Text To Speech service can't keep up with the playback rate to the caller, so audio might skip. transaction ID=a1b2c3d4e5",
  "id": "CWSMR0031W"
}
]
```

The object for each warning contains the following properties:

Property	Type	Description
message	String	The text of the warning message.
id	String	The unique message identifier.

#### Properties of the warnings\_and\_errors object

`max_response_milliseconds`

The `max_response_milliseconds` object contains the following properties:

Property	Type	Description
assistant	Number	The maximum round-trip latency (in milliseconds), calculated from all Watson Assistant requests related to the call.
text_to_speech	Number	The maximum time (in milliseconds) between when a text utterance is sent to the Text to Speech service and when the phone integration receives the first packet of synthesized audio. This value is calculated from all Text to Speech requests related to the call.
speech_to_text	Number	The maximum latency (in milliseconds) between when silence is detected in the caller's speech and when a final result from the Speech to Text service is received. This value is calculated from all Speech to Text recognition results related to the call.

#### Properties of the max\_response\_milliseconds object

`realtime_transport_network_summary`

When RTCP is enabled, the `realtime_transport_network_summary` object provides statistics for the inbound stream in the `inbound_stream` object and statistics for the outbound stream in the `outbound_stream` object.

The following example shows the structure of the `realtime_transport_network_summary` object:

```
$ "realtime_transport_network_summary": {
  "inbound_stream": {
    "maximum_jitter": 5,
    "average_jitter": 1,
    "packets_lost": 0,
    "packets_transmitted": 1000,
    "canonical_name": "user@example.com",
    "tool_name": "User SIP Phone"
  },
  "outbound_stream": {
    "maximum_jitter": 5,
    "average_jitter": 1,
    "packets_lost": 0,
    "packets_transmitted": 2000,
    "canonical_name": "voice.gateway@127.0.0.1",
    "tool_name": "IBM Voice Gateway/1.0.0.5"
  }
}
```

The object for each stream contains the following properties:

Properties	Type	Description
maximum_jitter	Number	The maximum jitter during the call.
average_jitter	Number	The average jitter, calculated over the duration of the call.
packets_lost	Number	An estimate of the number of packets that were lost during the call.
packets_transmitted	Number	An estimate of the total number of packets that were transmitted during the call.
canonical_name	String	A unique identifier for the sender of the stream, typically in @ format.
tool_name	String	The name of the application or tool where the stream originated. For the phone integration, the default is IBM Voice Gateway/ .

Properties of the realtime\_transport\_network\_summary object

# Segment event reference

The following tables show details of the events that Watson Assistant sends to Segment using the [Segment extension](#). These events appear as tables in your Segment warehouse, and as regular events in other Destinations.



**Note:** Only events generated using the Watson Assistant v2 API and associated with a user ID are included.

## Message Handled

Sent when the assistant completes handling of a message.

Property	Type	Description
accountId	String	The ID of the IBM account.
assistantId	String	The ID of the assistant.
browser	String	The browser that was used to send the message.
channel	String	The channel the customer used to send the message (for example, phone or chat).
device	String	The type of device that was used to send the message.
environment	String	The environment in which the message was handled (such as <code>draft</code> or <code>live</code> .)
language	String	The language of the assistant.
pageUrl	String	The URL of the web page from which the message was sent.
serviceInstanceId	String	The IBM Watson Assistant service instance.
sessionId	String	The ID of the session during which the message was handled.
skillsInvoked	String[]	An array of strings listing all skills that were invoked during handling of the message (for example, <code>main_skill</code> or <code>actions skill</code> ).

The following properties are included only for messages that were handled by an actions skill:

Property	Type	Description
action	String	The unique identifier of the action that was visited during handling of the message (for example, <code>action_202</code> ).
actionCompleted	Boolean	Whether the action completed during handling of the message.
actionCompletedReason	String	The reason the action completed (for example, <code>all_steps_done</code> or <code>fallback</code> .)
actionStarted	Boolean	Whether processing of the action started during handling of the message.

<code>actionTitle</code>	String	The title of the action that was visited during handling of the message (for example, I want to pay my bill).
<code>actionsVisited</code>	String[]	An array of strings listing the actions visited during handling of the message.
<code>fallbackReason</code>	String	The reason why the fallback action was visited (for example, escalated to human agent or no action matches).
<code>handler</code>	String	The name of any handler that was called.
<code>stepsVisited</code>	String[]	An array of strings listing the steps visited during handling of the message. Each step name is prefixed with the action name.
<code>subaction</code>	String	The name of any other action that was called by the action that was triggered by the message.

The following properties are included only for messages that were handled by a dialog skill:

Property	Type	Description
<code>branchExited</code>	Boolean	Whether the dialog branch was exited during handling of the message.
<code>branchExitedReason</code>	String	The reason the dialog branch was exited (for example, completed).
<code>nodesVisited</code>	String[]	An array of strings listing the dialog nodes visited during handling of the message. For each dialog node, the string specifies the node title (if any) or the node ID.

## Action Started

Sent when processing of an action begins.

Property	Type	Description
<code>accountId</code>	String	The ID of the IBM account.
<code>action</code>	String	The unique identifier of the action (for example, action_202).
<code>actionTitle</code>	String	The title of the action (for example, I want to pay my bill).
<code>actionCompleted</code>	Boolean	Whether the action completed during the same conversation turn.
<code>actionCompletedReason</code>	String	The reason the action completed (for example, all_steps_done or fallback.)
<code>assistantId</code>	String	The ID of the assistant.
<code>browser</code>	String	The browser that was used to send the message that triggered the action.
<code>channel</code>	String	The channel the customer used to send the message that triggered the action (for example, phone or chat).
<code>device</code>	String	The type of device that was used to send the message that triggered the action.

environment	String	The environment in which the action was started (such as <code>draft</code> or <code>live</code> .)
fallbackReason	String	The reason why the fallback action started (for example, escalated to human agent or no action matches).
handler	String	The name of any handler that was called.
language	String	The language of the assistant.
pageUrl	String	The URL of the web page from which the message that triggered the action was sent.
serviceInstanceId	String	The IBM Watson Assistant service instance.
sessionId	String	The ID of the session during which the message that started the action was sent.
skillsInvoked	String[]	An array of strings listing all skills that were invoked during handling of the message that started the action (for example, <code>main skill</code> or <code>actions skill</code> ).
stepsVisited	String[]	An array of strings listing the steps visited during processing of the action.
subaction	String	The name of any other action that the action called during processing.

## Action Completed

Sent when processing of an action ends.

Property	Type	Description
accountId	String	The ID of the IBM account.
action	String	The unique identifier of the action (for example, <code>action_202</code> ).
actionCompletedReason	String	The reason the action completed (for example, <code>all_steps_done</code> or <code>fallback</code> .)
actionStarted	Boolean	Whether the action was started during the same conversation turn.
actionTitle	String	The title of the action (for example, <code>I want to pay my bill</code> ).
assistantId	String	The ID of the assistant.
browser	String	The browser that was used to send the message that triggered the action.
channel	String	The channel the customer used to send the message that started the action (for example, <code>phone</code> or <code>chat</code> ).
device	String	The type of device that was used to send the message that triggered the action.
environment	String	The environment in which the action completed (such as <code>draft</code> or <code>live</code> .)

<code>fallbackReason</code>	String	The reason why the fallback action was called (for example, escalated to human agent or no action matches).
<code>handler</code>	String	The name of any handler that was called by the action.
<code>language</code>	String	The language of the assistant.
<code>pageUrl</code>	String	The URL of the web page from which the message that triggered the action was sent.
<code>serviceInstance</code>	String	The IBM Watson Assistant service instance.
<code>sessionId</code>	String	The ID of the session during which the message that started the action was sent.
<code>skillsInvoked</code>	String[]	An array of strings listing all skills that were invoked during handling of the message that started the action (for example, <code>main</code> skill or <code>actions</code> skill).
<code>stepsVisited</code>	String[]	An array of strings listing the steps visited during processing of the action.
<code>subaction</code>	String	The name of any other action that the action called during processing.

## Session Started

Sent when a new session is started.

**Note:** The v2 stateless API does not generate events for starting sessions.

Property	Type	Description
<code>accountId</code>	String	The ID of the IBM account.
<code>assistantId</code>	String	The ID of the assistant.
<code>browser</code>	String	The browser that was used to send the message that started the session.
<code>channel</code>	String	The channel that started the session (for example, phone or chat).
<code>device</code>	String	The type of device that was used to send the message that started the session.
<code>environment</code>	String	The environment in which the session was started (such as <code>draft</code> or <code>live</code> .)
<code>pageUrl</code>	String	The URL of the web page from which the message that started the session was sent.
<code>serviceInstance</code>	String	The IBM Watson Assistant service instance.
<code>sessionId</code>	String	The ID of the session.

# Filter query reference

The Watson Assistant service REST API offers powerful log search capabilities through filter queries. You can use the v2 /logs API `filter` parameter to search your skill log for events that match a specified query.

The `filter` parameter is a cacheable query that limits the results to those matching the specified filter. You can filter on various objects that are part of the JSON response model (for example, the user input text, the detected intents and entities, or the confidence score).

To see examples of filter queries, see [Examples](#).

For more information about the `/logs` `GET` method and its response model, refer to the [API Reference](#).

## Filter query syntax

The following example shows the general form of a filter query:

Location	Query operator	Term
<code>request.input.text</code>	<code>::</code>	<code>"IBM Watson"</code>

- The *location* identifies the field that you want to filter on (in this example, `request.input.text`).
- The *query operator*, which specifies the type of matching you want to use (fuzzy matching or exact matching).
- The *term* specifies the expression or value you want to use to evaluate the field for matching. The term can contain literal text and operators, as described in the [next section](#).

Filtering by intent or entity requires slightly different syntax from filtering on other fields. For more information, see [Filtering by intent or entity](#).

**Note:** The filter query syntax uses some characters that are not allowed in HTTP queries. Make sure that all special characters, including spaces and quotation marks, are URL encoded when sent as part of an HTTP query. For example, the filter `response_timestamp<2020-01-01` would be specified as `response_timestamp%3C2020-01-01`.

## Operators

You can use the following operators in your filter query.

Operator	Description
<code>:</code>	Fuzzy match query operator. Prefix the query term with <code>:</code> if you want to match any value that contains the query term, or a grammatical variant of the query term. Fuzzy matching is available for user input text, response output text, and entity values.
<code>::</code>	Exact match query operator. Prefix the query term with <code>::</code> if you want to match only values that exactly equal the query term.
<code>!:!</code>	Negative fuzzy match query operator. Prefix the query term with <code>!:!</code> if you want to match only values that do <i>not</i> contain the query term or a grammatical variant of the query term.
<code>&lt;=, &gt;=,</code> <code>&gt;, &lt;</code>	Comparison operators. Prefix the query term with these operators to match based on arithmetic comparison.

``	Escape operator. Use in queries that include control characters that would otherwise be parsed as operators. For example, !hello would match the text !hello.
""	Literal phrase. Use to enclose a query term that contains spaces or other special characters. No special characters within the quotation marks are parsed by the API.
~	Approximate match. Append this operator followed by a1 or 2 to the end of the query term to specify the allowed number of single-character differences between the query string and a match in the response object. For example, car~1 would match car, cat, or cars, but it would not match cats. This operator is not valid when filtering on log_id or any date or time field, or with fuzzy matching.
*	Wildcard operator. Matches any sequence of zero or more characters. This operator is not valid when filtering on log_id, language, request.context.system.assistant_id, workspace_id, request.context.metadata.deployment, or any date or time field.
( ), [ ]	Grouping operators. Use to enclose a logical grouping of multiple expressions using Boolean operators.
	Boolean or operator.
,	Boolean and operator.

## Filtering by intent or entity

Because of differences in how intents and entities are stored internally, the syntax for filtering on a specific intent or entity is different from the syntax used for other fields in the returned JSON. To specify an `intent` or `entity` field within an `intents` or `entities` collection, you must use the `:` match operator instead of a dot.

For example, this query matches any logged event where the response includes a detected intent named `hello`:

```
response.output.intents:intent::hello
```

Note the `:` operator in place of a dot (`intents:intent`)

Use the same pattern to match on any field of a detected intent or entity in the response. For example, this query matches any logged event where the response includes a detected entity with the value `soda`:

```
response.output.entities:value::soda
```

Similarly, you can filter on intents or entities sent as part of the request, as in this example:

```
request.input.intents:intent::hello
```

## Filtering by other fields

To filter on another field in the log data, specify the location as a path identifying the levels of nested objects in the JSON response from the /logs API. Use dots (`.`) to specify successive levels of nesting in the JSON data. For example, the location `request.input.text` identifies the user input text field as shown in the following JSON fragment:

```
$  "request": {
    "input": {
        "text": "Good morning"
    }
}
```

Filtering is not available for all fields. You can filter on the following fields:

- `assistant_id`

- `customer_id`
- `language`
- `request.context.global.system.user_id`
- `request.input.text`
- `request_timestamp`
- `response.context.global.system.user_id`
- `response.output.entities`
- `response.output.intents`
- `response_timestamp`
- `session_id`
- `skill_id`
- `snapshot`

Filtering on other fields is not currently supported.

## Examples

The following examples illustrate various types of queries using this syntax.

Description	Query
The date of the response is in the month of July 2020.	<code>response_timestamp&gt;=2020-07-01, response_timestamp&lt;2020-08-01</code>
The timestamp of the response is earlier than 2019-11-01T04:00:00.000Z.	<code>response_timestamp&lt;2019-11-01T04:00:00.000Z</code>
The message is labeled with the customer ID <code>my_id</code> .	<code>customer_id::my_id</code>
The message was sent to a specific assistant.	<code>assistant_id::dcd5c5ad-f3a1-4345-89c5-708b0b5ff4f7</code>
The user input text contains the word "order" or a grammatical variant (for example, orders or ordering).	<code>request.input.text:order</code>
An intent name in the response exactly matches <code>place_order</code> .	<code>response.output.intents:intent::place_order</code>
An entity name in the response exactly matches <code>beverage</code> .	<code>response.output.entities:entity::beverage</code>
No intent name in the response exactly matches <code>order</code> .	<code>response.intents:intent::!order</code>

The user input text does not contain the word "order" or a grammatical variant.

```
request.input.text: !order
```

The user input text contains the string !hello.

```
request.input.text: !hello
```

The user input text contains the string IBM Watson.

```
request.input.text: "IBM Watson"
```

The user input text contains a string that has no more than 2 single-character differences from Watson.

```
request.input.text: Watson~2
```

The user input text contains a string consisting of comm, followed by zero or more additional characters, followed by s.

```
request.input.text: comm*
```

An intent name in the response exactly matches either hello or goodbye.

```
response.output.intents:intent::(hello|goodbye)
```

An intent name in the response exactly matches order, and an entity name in the response exactly matches beverage.

```
[response.output.intents:intent::order, response.output.entities:entity::beverage]
```

## Filtering v1 logs

If your application is still using the v1 API, you can query and filter logs using the v1 /logs method. The filtering syntax is the same, but the structure of v1 logs and message requests is different. For more information, see [API Reference](#).

With the v1 /logs API, you can filter on the following fields:

- language
- meta.message.entities\_count
- request.context.metadata.deployment
- request.context.system.assistant\_id
- request.input.text
- response.context.conversation\_id
- response.entities
- response.input.text
- response.intents
- response.top\_intent
- workspace\_id

Filtering on other fields is not currently supported.

# FAQs for IBM Watson® Assistant

Find answers to frequently-asked questions and quick fixes for common problems.

## FAQs about the new IBM Watson® Assistant experience

### **What is the new IBM Watson® Assistant experience?**

The new Watson Assistant is an improved way to build, publish, and improve virtual assistants. In the new experience, you use actions to build conversations. Actions are a simple way for anyone, developer or not, to create assistants. For more information, see the [Getting Started guide](#) or the [documentation](#) for the new experience.

### **Why can't I see the assistants I made with classic Watson Assistant in the new experience?**

The new Watson Assistant is a clean slate in the same IBM Cloud instance as your classic experience. Assistants you created in one experience don't appear in the other. However, you can switch back and forth between experiences without losing any work. For more information, see [Switching the experience](#).

### **What happens when I switch between the classic and new Watson Assistant experiences?**

The assistants you create in one experience don't transfer to the other. However, you can switch experiences, return to your work, and create or use assistants. You won't lose anything by switching. Changing experiences doesn't affect other users working in the same instance. For more information, see [Switching the experience](#).

### **Is the classic Watson Assistant experience going away?**

IBM has no plans to discontinue the classic Watson Assistant experience. However, we encourage you to explore the benefits and capabilities in the new Watson Assistant. For more information, see the [Getting Started guide](#) or the [documentation](#) for the new experience.

### **Where are the search skill and channel integrations in the new Watson Assistant experience?**

In the left navigation, click **Integrations** . On the Integrations page, you can add search, channel, and extension integrations to your assistant. For more information, see [Adding integrations](#).

### **Where is the Assistant ID found in the new product experience?**

The assistant ID can be found in **Assistant settings**.

In **Assistant settings**, the assistant ID is in the **Details** section.

## What do the draft and live tags mean?

A **Draft** tag indicates that the information is linked to your draft environment, which means that you can preview these updates but they are not visible to your users. A **Live** tag indicates that the information is linked to your live environment, which means that the content is available to your users to interact with.

For more information on environments, see [Environments](#).

## Why can't I log in?

If you are having trouble logging in to a service instance or see messages about tokens, such as `unable to fetch access`

token or 400 bad request - header or cookie too large, it might mean that you need to clear your browser cache. Open a private browser window, and then try again.

- If accessing the page by using a private browsing window fixes the issue, then consider always using a private window or clear the cache of your browser. You can typically find an option for clearing the cache or deleting cookies in the browser's privacy and security settings.
- If accessing the page by using a private browsing window doesn't fix the issue, then try deleting the API key for the instance and creating a new one.

## Why am I being asked to log in repeatedly?

---

If you keep getting messages, such as you are getting redirected to login, it might be due to one of the following things:

- The Lite plan you were using has expired. Lite plans expire if they are not used within a 30-day span. To begin again, log in to IBM Cloud and create a new service instance of Watson Assistant.
- An instance is locked when you exceed the plan limits for the month. To log in successfully, wait until the start of the next month when the plan limit totals are reset.

## Why don't I see the Analytics page?

---

To view the **Analytics** page, you must have a service role of Manager and a platform role of at least Viewer. For more information about access roles and how to request an access role change, see [Managing access to resources](#).

## Why am I unable to view the API details, API key, or service credentials?

---

If you cannot view the API details or service credentials, it is likely that you do not have Manager access to the service instance in which the resource was created. Only people with Manager access to the instance can use the service credentials.

## Can I export the user conversations from the Analytics page?

---

You cannot directly export conversations from the conversation page. You can, however, use the /logs API to list events from the transcripts of conversations that occurred between your users and your assistant. For more information, see the [API reference](#).

## Can I change my plan to a Lite plan?

---

No, you cannot change from a Trial, Plus, or Standard plan to a Lite plan. And you cannot upgrade from a Trial to a Standard plan.

## How do I create a webhook?

---

To define a webhook and add its details, go to the **Live environment** page and open the **Environment settings** page. From the **Environment settings** page, click **Webhooks > Pre-message webhook**. From this page, you can add details about your webhook. For more information, see [Making a call before processing a message](#).

## Can I have more than one entry in the URL field for a webhook?

---

No, you can define only one webhook URL for an action. For more information, see [Defining the webhook](#).

## Is there a range of IP addresses that are being used by a webhook?

---

Unfortunately, the IP address ranges from which Watson Assistant may call a webhook URL are subject to change, which in turn prevent using them in any static firewall configuration. Please use the https transport and specify an authorization header to control access to the webhook.

## **What do I do if the training process seems stuck?**

---

If the training process gets stuck, first check whether there is an outage for the service by going to the [Cloud status page](#). You can start a new training process to stop the current process and start over.

## **How do I see my monthly active users in Watson Assistant?**

---

To see your monthly active users (MAU) do the following:

1. Sign in to <https://cloud.ibm.com>
2. Click on the **Manage** menu, then choose **Billing and usage**.
3. Click on **Usage**.
4. For Watson Assistant, select **View Plans**.
5. Under Time Frame, select the month you need.
6. Select your Plus plans or Plus Trial plans to see monthly active users and the API calls.

# Getting help

Get help with solving issues that you encounter while using the product.

Use these resources to get answers to your questions:

- For answers to frequently asked questions, see the [FAQ](#).
- Click the **Learning center** link that is displayed in the header of the skill pages to find helpful product tours. The tours guide you through the steps to follow to complete a range of tasks, from adding your first intent to a dialog skill to enhancing the conversation in an actions skill. The *Additional resources* page has links to relevant documentation topics and how-to videos. You can search the resource link titles to find what you're looking for quickly.
- Find answers to common questions or ask questions where experts and other community members can answer by visiting the [Watson Assistant Community forum](#).

If your service plan covers it, you can get help by creating a case from [IBM Cloud Support](#).

# Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

IBM Watson® Assistant uses standard Windows navigation keys.

## **Accessibility features in the product documentation**

---

Accessibility features help people with a physical disability, such as restricted mobility or limited vision, or with other special needs, use information technology products successfully.

The accessibility features in this product documentation allow users to do the following:

- Use screen-reader software and digital speech synthesizers to hear what is displayed on the screen. Consult the product documentation of the assistive technology for details on using assistive technologies with HTML-based information.
- Use screen magnifiers to magnify what is displayed on the screen.
- Operate specific or equivalent features by using only the keyboard.

The documentation content is published in the IBM Cloud Docs site. For information about the accessibility of the site, see [Accessibility features for IBM Cloud](#).

## Browser support

The Watson Assistant application requires the same level of browser software as is required by IBM Cloud. For more information, see [IBM Cloud prerequisites](#).

For information about the web browsers that are supported by the web chat integration, see [Browser support](#).

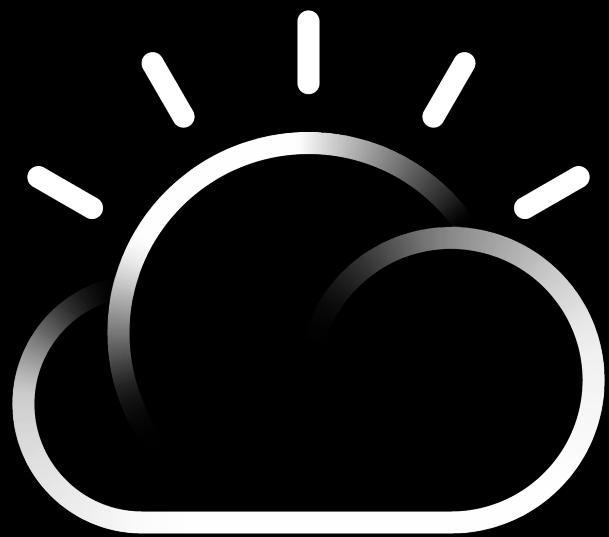
# Terms and notices

See [IBM Cloud Terms and Notices](#) for information about the terms of service.

US Health Insurance Portability and Accountability Act (HIPAA) support is available with *Enterprise with Data Isolation* plans that are hosted in the Washington, DC location created on or after 1 April 2019. For more information, see [Enabling HIPAA support for your account](#).

To learn more about service terms and data security, read the following information:

- [Service terms](#) (Search for the Watson Assistant offering)
- [Data Processing and Protection Datasheet](#)
- [IBM Cloud Data security and privacy](#)



IBM Cloud